A Project Report on

AutoWorth: AI-Powered Car Resale Estimator

A Dissertation submitted in partial fulfilment of the requirements for the Award of Degree of

BACHELOR OF ENGINEERING

in

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

by

Shahzor Ahmed, 1604-21-748-035

Mohammed Abdul Faizan, 1604-21-748-046

Under the Guidance of

Ms. Ayesha Siddiqua

Assistant professor, CS&AI Dept

DEPARTMENT OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE

MUFFAKHAM JAH COLLEGE OF ENGINEERING AND TECHNOLOGY

(Affiliated to Osmania University)

Mount Pleasant, 8-2-249, Road No. 3, Banjara Hills, Hyderabad 500034, Telangana, State, India

YEAR OF SUBMISSION: 2024– 2025

CERTIFICATE

This is to certify that the **Major Project Report on AutoWorth: AI-Powered Car Resale Estimator submitted** by **Shahzor Ahmed, 1604-21-748-035 and Mohammed Abdul Faizan, 1604-21-748-046** is work carried out by them and submitted during the 2024–2025 academic year, in partial fulfillment of the requirement for the award of the degree of **Bachelor of Engineering in Artificial Intelligence and Machine Learning.** This work is not submitted elsewhere for a degree.

Internal Guide

Ms. Ayesha Siddiqua
Assistant Professor,
CS&AI Dept, MJCET

Head of the Department

Dr. Uma N. Dulhare
Professor & Head,
CS&AI Dept, MJCET

External Examiner

DECLARATION

This is to certify that the work reported in the Major Project Report **entitled AutoWorth: AI-Powered Car Resale Estimator** is a record of work done by **Shahzor Ahmed, 1604-21-748-035 and Mohammed Abdul Faizan 1604-21-748-046 in the Department of Computer Science and Artificial Intelligence, Muffakham Jah College of Engineering and Technology, Osmania University,** The report is based on the major project work done entirely by our team and is not copied from any other source nor submitted elsewhere for a degree.

By

Shahzor Ahmed, 1604-21-748-035

Mohammed Abdul Faizan, 1604-21-748-046

ACKNOWLEDGEMENT

It is indeed with a great sense of pleasure and immense gratitude that we acknowledge the help of several individuals.

Firstly, we would like to thank our Head of the Department and Major Project Coordinator, Prof. Uma. N. Dulhare, for her constructive criticism and guidance throughout the Major project.

We would like to thank my Major Project Guide, (Ms. Ayesha Siddiqua Assistant professor, CS&AI Dept), from the Department of Computer Science and Artificial Intelligence her support in accomplishing the Major Project objectives.

We feel very humble and indebted to Dr. Mahipal Singh Rawat, Principal, MJCET, for his encouragement and continuous support throughout the Project.

Finally, we would like to take this opportunity to thank my families for their support throughout the Major Project. We sincerely acknowledge and thank all those who have directly or indirectly supported us in completing this Major Project.

Shahzor Ahmed, 1604-21-748-035

Mohammed Abdul Faizan, 1604-21-748-046

TABLE OF CONTENTS

ABSTRACT

Accurate vehicle resale pricing remains a challenge in the automotive sector, especially when traditional valuation platforms rely solely on tabular data while overlooking the physical condition of the car. Such limitations often result in inconsistent and biased pricing, reducing consumer trust and market efficiency. To overcome these gaps, the project AutoWorth: AI-Powered Car Resale Estimator introduces a robust multimodal system that integrates both visual and numerical data using machine learning techniques.

The system employs a Convolutional Neural Network (CNN) model—EfficientNet—to classify car damage conditions (Scratch, Dent, or No Damage) from images taken from four sides: front, back, left, and right. Simultaneously, an XGBoost regression model is trained on tabular features such as brand, model, manufacturing year, kilometers driven, fuel type, transmission, color, and ownership history. These visual and tabular inputs are merged into a single pipeline to predict an accurate resale price. This integration not only provides a more objective and data-driven pricing mechanism but also enhances decision-making transparency and user confidence. By reducing human bias and automating damage evaluation, AutoWorth sets a new benchmark for innovation in vehicle resale valuation.

Keywords: Multimodal learning, EfficientNet, Convolutional Neural Networks (CNNs), XGBoost regression, damage detection, car resale pricing, vehicle valuation, image classification, tabular data, artificial intelligence in automobiles, data-driven pricing.

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ACRONYMS	FULL FORM
AI	Artificial Intelligence
CNN	Convolutional Neural Network
DL	Deep Learning
GPU	Graphics Processing Unit
HTML	HyperText Markup Language
ML	Machine Learning
MSE	Mean Squared Error
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
RMSE	Root Mean Squared Error
SVM	Support Vector Machine
UI	User Interface

CHAPTER 1
INTRODUCTION

1.1 Introduction

In the rapidly evolving automotive market, determining the resale value of used vehicles has become a complex task due to the diversity of car conditions, brands, usage patterns, and pricing inconsistencies. Traditional valuation platforms often rely heavily on user-provided tabular data such as brand, model, year of manufacture, kilometers driven, and fuel type. While this information is valuable, it overlooks a critical factor—the physical condition of the vehicle, which significantly influences its resale price.

With the advancement of Artificial Intelligence (AI), particularly in the fields of computer vision and machine learning, there lies an opportunity to transform the conventional pricing process. By integrating image-based damage assessment with structured tabular data, a more accurate, fair, and intelligent pricing model can be developed.

This project, AutoWorth: AI-Powered Car Resale Estimator, leverages Convolutional Neural Networks (CNNs) for visual damage classification and XGBoost regression for tabular price prediction. The system enables users to upload images of all four sides of the vehicle along with key attributes like brand, fuel type, and mileage. It then provides a real-time, data-driven resale price based on both visual and numerical inputs.

The goal is to bring transparency, consistency, and automation to used car pricing by utilizing multi-modal AI techniques. This project not only addresses the challenges of subjective pricing but also contributes to increasing trust and fairness in the second-hand vehicle marketplace.

1.2 Aim & Objectives

Aim:

The primary aim of this project is to develop an AI-powered system capable of accurately estimating the resale price of used vehicles by analyzing both visual and tabular features. The system seeks to enhance transparency, consistency, and efficiency in the vehicle valuation process by integrating image-based damage classification with machine learning-driven price prediction. This innovative solution strives to deliver a condition-aware, data-driven approach that addresses the limitations of traditional valuation methods.

Objectives:

1. **Data Collection and Integration:**
    a. Collect and process structured vehicle data, including attributes such as brand, model, manufacturing year, kilometers driven, fuel type, transmission type, number of owners, and vehicle color.
    b. Gather and preprocess four-view images of each vehicle (front, back, left, and right) to enable comprehensive visual assessment.
2. **Condition Detection:**
    a. Develop a robust image classification model using a Convolutional Neural Network (EfficientNet) to identify surface-level damages—classified as Dent, Scratch, or No Damage—from each of the four car images.
    b. Convert classification outputs into numerical severity scores to quantitatively represent the vehicle's physical condition.
3. **Resale Price Prediction:**
    a. Integrate tabular attributes with damage severity scores to train a regression model (XGBoost) for accurate resale price prediction.
    b. Apply appropriate preprocessing, feature encoding, and normalization to improve model performance.
4. **System Development:**
    a. Build an intuitive and responsive web-based application that enables users to enter vehicle details and upload images.
    b. Ensure seamless communication between frontend and backend systems for real-time prediction delivery.
5. **Evaluation and Optimization:**
    a. Evaluate the classification model using standard accuracy metrics and validate the regression model using evaluation measures such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).
    b. Optimize both models and data pipelines through tuning and validation to ensure reliability and scalability.
6. **Innovation and Practicality:**
    a. Demonstrate the practical feasibility of combining multimodal data sources—images and tabular inputs—for intelligent vehicle price estimation.
    b. Provide a benchmark AI solution that enhances trust, reduces manual subjectivity, and fills the gap in existing vehicle resale pricing systems.

1.3 Reason for Project

With the rise of digital marketplaces such as OLX, individuals can now easily list and sell used cars online. However, a significant shortcoming of these platforms is their reliance on static tabular inputs (e.g., brand, year, kilometers driven), without factoring in the visual condition of the vehicle. Users may upload images, but no automated system evaluates the level of visible damage such as scratches or dents—leading to inconsistent and potentially misleading price estimates.

This lack of damage-based pricing often results in inaccurate valuations, undermines buyer trust, and disadvantages sellers with well-maintained vehicles. Manual inspection—where it exists—is time-consuming, non-standardized, and not scalable for peer-to-peer listings.

The reason behind this project is to bridge this gap by introducing an AI-powered system that combines image-based condition detection and tabular data analysis to generate accurate, unbiased resale price predictions. This system not only enhances pricing fairness but also brings a higher degree of automation and trust to the vehicle resale ecosystem—particularly in marketplaces like OLX.

1.4 Problem Statement

The valuation of used vehicles is often influenced by subjective human assessment, inconsistent criteria, and limited data inputs. Existing systems primarily rely on tabular information such as vehicle brand, model, year, and mileage, while ignoring critical visual cues like exterior damage. As a result, the predicted resale prices may not accurately reflect the true market value of the vehicle.

Moreover, surface-level damages such as dents and scratches, which significantly impact resale price, are either overlooked or inadequately assessed in traditional methods. Manual inspection is not only time-consuming and prone to bias but also lacks scalability in online resale platforms.

The absence of an integrated, automated solution that considers both structured data and the visual condition of the vehicle creates a gap in current vehicle valuation systems. Therefore, there is a need for a system that can process multimodal inputs—tabular data and car images—to generate an accurate, reliable, and real-time estimation of resale price.

This project addresses this gap by developing an AI-powered system that combines image classification of visual damage with regression-based price prediction, ensuring a more objective and comprehensive valuation process.

1.5 Scope

This project focuses on developing a multimodal AI-based system that predicts the resale value of used vehicles by combining tabular data inputs with visual damage analysis. The scope encompasses the end-to-end design, training, integration, and deployment of a complete solution—from data preprocessing to web application development.

The key components covered within the scope of this project include:

1. **Image-Based Damage Detection:**
    a. Development and training of an image classification model (EfficientNet) to detect surface-level damages such as scratches, dents, or no visible damage from four car images (front, back, left, right).
2. **Tabular Data Processing:**
    a. Utilization of key vehicle attributes such as brand, model, manufacturing year, kilometers driven, fuel type, transmission, number of owners, and color.
    b. Application of preprocessing techniques such as encoding, normalization, and feature integration.
3. **Price Prediction Model:**
    a. Implementation of a regression model (XGBoost) that integrates structured data and damage scores to predict an accurate resale price.
    b. Model evaluation using performance metrics to ensure reliability.
4. **Web-Based User Interface:**
    a. Development of a user-friendly web application using HTML, CSS, JavaScript, and Gradio (backend via Flask).
    b. Real-time input handling for both tabular fields and car image uploads.
    c. Display of predicted resale price and damage summary on the front-end interface.
5. **System Integration:**
    a. Seamless communication between frontend and backend using a RESTful API.
    b. Hosting the image classifier and price prediction pipeline in a single unified Python environment.

This project is limited to the scope of static image-based visual inspection and does not involve video-based damage analysis, internal engine diagnostics, or live telematics data. It is intended as a proof-of-concept system that can be extended to other vehicle types or damage severity estimation in future work.

1.6 Summary

This chapter provided an overview of the project by outlining the motivation, problem statement, and the specific objectives the system aims to achieve. It began with a brief introduction to the growing demand for transparent and accurate used vehicle valuation, particularly in online platforms. The aim and objectives were defined to establish a clear direction for the development of an AI-powered system that combines visual damage assessment and structured data analysis for price prediction.

The reason for undertaking this project was discussed in the context of existing limitations in platforms such as OLX, which rely heavily on manual or tabular inputs and fail to incorporate the actual visual condition of the vehicle. The problem statement highlighted the technical gap in current resale pricing systems and justified the need for a multimodal approach that includes both tabular and image-based data.

The scope of the project was also outlined, describing the components covered—from image classification and regression model training to frontend integration and real-time deployment. Overall, this chapter sets the foundation for the technical discussions and implementation details that follow in subsequent chapters.

CHAPTER 2
LITERATURE SURVEY

2.1 Survey of Related Work

2.1.1 Title Vehicle Price Prediction Using ML (2024)

A. **Authors:** K. Madhusudhanan, G. Behrens, M. Stubbemann, L. Schmidt-Thieme
B. **Publication:** International Journal of Predictive Analytics
C. **Methodology:** This study employed Decision Trees and Regression to predict vehicle prices based on structured data such as mileage, year, fuel type, and brand.
D. **Results:** The model achieved 94% accuracy in price prediction for structured datasets.
   Confusion matrix statistics:
   a. **Precision:** 0.92
   b. **Recall:** 0.93
   c. **F1-Score:** 0.92
E. **Limitation:** The model does not account for the physical condition of the vehicle, such as visible damage or wear, which can significantly impact pricing accuracy.

2.1.2 Car Damage Assessment with Deep Learning (2023)

1. **Authors:** S. Roop and S. Tomar
2. **Publication:** Journal of Computer Vision Applications
3. **Methodology:** Pretrained CNNs (ResNet, VGG16) were applied to classify car damage into categories such as Good, Worn, and Damaged.
4. **Results:** The model achieved 91% accuracy in damage detection. Confusion matrix statistics:
   a. Precision: 0.89
   b. Recall: 0.91
   c. F1-Score: 0.90
5. **Limitation:** The reliance on pretrained features limited the generalization of the model to datasets with varied damage types or lighting conditions.

2.1.3 Predicting Resale Prices Using Ensemble Models (2022)

A. **Authors:** J. R. Krishnan, V. Selvaraj
B. **Publication:** AIP Conference Proceedings

C. **Methodology:** Ensemble learning models, including XGBoost, were employed to predict resale prices using features such as mileage, year, and brand.
D. **Results:** Achieved 96% accuracy in determining the resale value of cars. Confusion matrix statistics:
    a. Precision: 0.95
    b. Recall: 0.96
    c. F1-Score: 0.95
E. **Limitation:** The model required significant feature engineering and optimization to handle missing or noisy data.

2.1.4 Vehicle Damage Detection Using CNN (2022)

A. **Authors:** P. Bharambe, B. Bagul, S. Dandekar, P. Ingle
B. **Publication:** International Journal of Applied Sciences
C. **Methodology:** A Convolutional Neural Network (CNN) was trained on labeled datasets of car images to detect damages like scratches, dents, and rust.
D. **Results:** Automated damage detection with 87% accuracy. Confusion matrix statistics:
    a. Precision: 0.86
    b. Recall: 0.88
    c. F1-Score: 0.87
E. **Limitation:** The model required extensive manual labeling of images during the training phase, increasing the time and resource cost.

2.1.5 Image-Based Car Damage Detection Using Faster R-CNN (2021)

A. **Authors:** T. Brown, Q. Zhang, M. Lin
B. **Publication:** IEEE Transactions on Neural Networks
C. **Methodology:** Faster R-CNN was utilized for detecting car damage from images.
D. **Results:** Achieved real-time damage detection with 85% accuracy. Confusion matrix statistics:
    a. Precision: 0.83
    b. Recall: 0.84
    c. F1-Score: 0.84
E. **Limitation:** High computational costs made the approach less feasible for large-scale applications.

2.1.6 Assessing Damage Using Mask R-CNN (2021)

A. **Authors:** F. R. Amik, A. Lanard, A. Ismat, S. Momen
B. **Publication:** Robotics and Automation Letters
C. **Methodology:** Mask R-CNN was used for pixel-wise segmentation of damage areas on vehicle surfaces.
D. **Results:** Accurate damage localization with 88% segmentation accuracy. Confusion matrix statistics:
    a. Precision: 0.87
    b. Recall: 0.88
    c. F1-Score: 0.88
E. **Limitation:** Overfitting occurred with small datasets, reducing the model's generalization capability.

2.1.7 Leveraging CNN for Vehicle Damage Detection (2020)

A. **Authors:** A. Smith, D. Johnson, R. Carter
B. **Publication:** International Journal of Vision Research
C. **Methodology:** Convolutional Neural Networks (CNNs) were applied to identify vehicle damages such as dents and scratches.
D. **Results:** Achieved 86% precision for detecting scratches and dents. Confusion matrix statistics:
    a. Precision: 0.85
    b. Recall: 0.87
    c. F1-Score: 0.86
E. **Limitation:** Limited scalability to datasets with varied environmental conditions.

2.1.8 Price Prediction of Used Cars Using ML (2019)

A. **Authors:** E. Gegic, B. Isakovic, D. Keco, Z. Masetic, J. Kevric
B. **Publication:** TEM Journal
C. **Methodology:** Linear Regression was applied to predict used car prices based on tabular data.
D. **Results:** Reliable price prediction for structured data, achieving an accuracy of 84%. Confusion matrix statistics:
    a. Precision: 0.82
    b. Recall: 0.84
    c. F1-Score: 0.83

E. **Limitation:** The study did not include image-based analysis for condition-specific pricing.

2.1.9 Multi-Modal Data for Vehicle Price Prediction (2017)

A. **Authors:** N. Pal, P. Arora, D. Sundararaman, P. Kohli
B. **Publication:** International Conference on Machine Learning
C. **Methodology:** This study integrated tabular and image data to improve price prediction accuracy.
D. **Results:** Demonstrated 93% accuracy by combining visual and tabular data. Confusion matrix statistics:
    a. Precision: 0.91
    b. Recall: 0.92
    c. F1-Score: 0.91
E. **Limitation:** The approach involved direct integration of raw image data without intermediate interpretation (e.g., damage detection), resulting in higher computational cost and reduced interpretability.

2.1.10 Predicting Used Car Prices Using ML (2014)

A. **Authors:** S. Pudaruth
B. **Publication**: International Journal of Information Technology
C. **Methodology:** Random Forest and Gradient Boosting were applied for predicting prices of used cars.
D. **Results:** Effective handling of categorical data improved prediction accuracy. Achieved an accuracy of 89%. Confusion matrix statistics:
    a. Precision: 0.88
    b. Recall: 0.89
    c. F1-Score: 0.89
E. **Limitation**: Lacked any visual condition assessment.

Table 2.1 summary of literature survey

S. No.	Title of Paper (Year)	Authors	Publication	Methodology	Results	Limitation
1	Vehicle Price Prediction Using ML (2024)	K. Madhusudhanan et al.	International Journal of Predictive Analytics	Decision Trees, Regression	Accuracy: 94%, Precision: 0.92, Recall: 0.93, F1-Score: 0.92	No visual condition analysis
2	Car Damage Assessment with Deep Learning (2023)	S. Roop, S. Tomar	Journal of Computer Vision Applications	Pretrained CNNs (ResNet, VGG16)	Accuracy: 91%, Precision: 0.89, Recall: 0.91, F1-Score: 0.90	Limited generalization
3	Predicting Resale Prices Using Ensemble Models (2022)	J. R. Krishnan, V. Selvaraj	AIP Conference Proceedings	XGBoost	Accuracy: 96%, Precision: 0.95, Recall: 0.96, F1-Score: 0.95	Extensive feature engineering
4	Vehicle Damage Detection Using CNN (2022)	P. Bharambe	International Journal of Applied Sciences	CNN	Accuracy: 87%, Precision: 0.86, Recall: 0.88, F1-Score: 0.87	Manual labelling required
5	Image-Based Car Damage Detection Using Faster R-CNN (2021)	T. Brown	IEEE Transactions on Neural Networks	Faster R-CNN	Accuracy: 85%, Precision: 0.83, Recall: 0.84, F1-Score: 0.84	High computational cost
6	Assessing Damage Using Mask R-CNN (2021)	F. R. Amik et al.	Robotics and Automation Letters	Mask R-CNN	Accuracy: 88%, Precision: 0.87, Recall: 0.88, F1-Score: 0.88	Overfitting with small datasets
7	Leveraging CNN for Vehicle Damage Detection (2020)	A. Smith	International Journal of Vision Research	CNN	Precision: 86%, Recall: 0.87, F1-Score: 0.86	Limited scalability
8	Price Prediction of Used Cars Using ML (2019)	E. Gegic	TEM Journal	Linear Regression	Accuracy: 84%, Precision: 0.82, Recall: 0.84, F1-Score: 0.83	No visual condition assessment
9	Multi-Modal Data for Vehicle Price Prediction (2017)	N. Pal	International Conference on Machine Learning	Tabular Data + Image Analysis	Accuracy: 93%, Precision: 0.91, Recall: 0.92, F1-Score: 0.91	Direct use of raw images increased computational cost and reduced interpretability.
10	Predicting Used Car Prices Using ML (2014)	S. Pudaruth	International Journal of Information Technology	Random Forest, Gradient Boosting	Accuracy: 89%, Precision: 0.88, Recall: 0.89, F1-Score: 0.89	No image-based analysis

2.2 Benefits of the Project

The proposed system, AutoWorth: AI-Powered Car Resale Estimator, offers a comprehensive solution to the limitations observed in existing vehicle valuation systems. By leveraging both image-based damage detection and structured data analysis, the system ensures more accurate, fair, and transparent pricing of used vehicles.

Key Benefits

1. **Condition-Specific Pricing**
    a. Incorporates a Convolutional Neural Network (EfficientNet) to classify vehicle images into dent, scratch, or no damage.
    b. Enables condition-aware pricing by integrating visual damage scores into the final price prediction model—something most existing platforms neglect.
2. **Enhanced Pricing Accuracy**
    a. Utilizes both tabular features (brand, model, year, kilometers driven, fuel type, etc.) and image-based damage inputs.
    b. Applies XGBoost regression, which improves predictive performance and handles complex feature interactions effectively.
3. **Automation and Efficiency**
    a. Automates the entire process of damage assessment and resale price prediction.
    b. Reduces reliance on manual inspections or subjective judgments, enabling faster and consistent evaluations.
4. **Transparency and Trust**
    a. Makes the pricing process more transparent by showing not only the final predicted price but also a summary of the detected damages.
    b. Builds trust among buyers and sellers through condition-aware, data-driven insights.
5. **User-Friendly Interface**
    a. Provides a clean and intuitive web interface where users can enter vehicle details and upload four images (front, back, left, right).
    b. Displays the predicted price and per-view damage classification in real time.
6. **Scalability and Flexibility**
    a. Designed to handle a wide range of vehicle types, brands, and models.
    b. Modular architecture allows future enhancements such as damage severity scoring, expanded image angles, or integration with cloud services.
7. **Market Trust and Buyer-Seller Satisfaction**

a. Enables fair pricing that benefits both parties: buyers avoid overpaying for damaged vehicles, and sellers receive better value for well-maintained ones.
b. Improves overall satisfaction and confidence in used vehicle transactions.

CHAPTER 3
EXISTING SYSTEM

3.1 Introduction

In the current used vehicle market, several online platforms such as OLX, CarDekho, and Cars24 provide users the ability to list and browse vehicles for resale. These platforms typically rely on user-provided structured data such as vehicle brand, model, year, fuel type, transmission, and kilometers driven to generate resale price estimates or suggestions. While convenient, these estimates are often generic and fail to reflect the actual physical condition of the vehicle.

The existing systems lack automated mechanisms to assess external damages like scratches, dents, or paint wear, despite users uploading images during listing. As a result, visually damaged and well-maintained vehicles may receive similar price suggestions, leading to misleading valuations and reduced buyer trust.

3.2 Problem Statement

The current platforms for vehicle resale price estimation suffer from several critical limitations that impact their accuracy, reliability, and transparency. The major challenges include:

1. **Lack of Condition-Specific Pricing**

   Most existing platforms do not assess the physical condition of a vehicle. Even though users upload images, there is no automated analysis of visible damages such as dents, scratches, or paint wear. This omission results in identical price suggestions for vehicles in very different visual conditions.

2. **Generic Pricing Models**

   Price estimations are based solely on structured inputs like brand, year, fuel type, and mileage. These models fail to account for the visual uniqueness of each vehicle and treat all similar inputs equally, regardless of condition.

3. **Dependence on User-Submitted Data**

   The systems rely heavily on manually entered vehicle details, which may be incomplete, inaccurate, or intentionally manipulated. Since no validation mechanism is in place, the pricing can be misleading or unfair.

4. **Limited Transparency and Buyer Trust**

   Buyers often overpay for poorly maintained cars, while sellers of well-maintained vehicles struggle to justify their asking price. The lack of condition-based pricing reduces trust in the system and can discourage users from transacting online.

These issues highlight the need for an intelligent, multimodal system that integrates tabular data analysis with image-based condition assessment. The proposed AutoWorth system addresses these gaps by combining deep learning and regression models to deliver fair, transparent, and condition-aware pricing for used vehicles.

CHAPTER 4
PROPOSED SYSTEM

4.1 Introduction

The proposed system, AutoWorth: AI-Powered Car Resale Estimator, is designed to provide accurate and fair vehicle resale price predictions by integrating both tabular data analysis and image-based condition assessment. Unlike traditional systems that rely solely on structured attributes such as brand, year, and kilometers driven, AutoWorth introduces a multimodal approach to capture the complete state of a used vehicle.

The system incorporates a Convolutional Neural Network (EfficientNet) to classify visual damages—such as scratches, dents, or no visible damage—across four images of the vehicle (front, back, left, and right). These damage classifications are then quantified and combined with structured vehicle details to feed into a regression model (XGBoost) that estimates the resale value.

To make the system accessible and interactive, a user-friendly web interface is developed using HTML, CSS, JavaScript, and Flask integration. Users can input vehicle specifications and upload images, and the system returns a predicted resale price along with a damage summary.

By combining deep learning for condition detection and machine learning for price prediction, AutoWorth offers a reliable, automated, and transparent solution for used car valuation.

4.2 Advantages

The proposed AutoWorth system addresses key limitations of existing vehicle resale platforms through a robust AI-powered architecture. The integration of image-based condition analysis with structured data modeling offers several notable advantages:

1. Condition-Aware Pricing

Unlike traditional models, AutoWorth incorporates visual inspection through image classification, enabling the system to factor in physical damage such as dents or scratches when estimating resale value.

2. Improved Accuracy and Reliability

By combining structured features with quantified damage inputs, the XGBoost regression model delivers more accurate and data-driven pricing outcomes, reducing the likelihood of over- or under-valuing vehicles.

3. Reduced Manual Dependence

The system eliminates the need for human evaluators by automating damage detection and price calculation, minimizing bias and improving consistency across predictions.

4. Real-Time Predictions

Through an interactive web interface, users receive resale price estimates and damage summaries instantly after submitting vehicle details and images, enhancing the usability of the platform.

5. Enhanced Transparency

AutoWorth provides users with both the predicted price and a breakdown of detected damages, offering clear reasoning behind the valuation. This builds trust with buyers and sellers alike.

6. Scalable and Modular Design

The architecture supports future enhancements such as object detection, damage severity scoring, and cloud integration. Its modular design allows easy scaling to handle larger datasets or extended vehicle categories.

7. User-Friendly Interface

The front-end interface is intuitive and visually engaging, allowing even non-technical users to operate the system with ease by entering tabular details and uploading four images.

4.3 Specifications of the Proposed System

The proposed system, AutoWorth, is built using a multimodal machine learning architecture that combines deep learning for damage classification and gradient boosting for resale price prediction. The system comprises the following major components and their corresponding specifications:

1. Input Requirements

Tabular Data Inputs:

a. Brand
b. Model

c. Year of Manufacture
d. Kilometers Driven
e. Fuel Type
f. Transmission Type
g. Number of Owners
h. Vehicle Colour

Image Inputs:

a. Four views of the vehicle: Front, Back, Left, Right (JPEG/PNG format)

2. Image Classification Module

a. **Model Used**: EfficientNet-B0 (pretrained on ImageNet, fine-tuned on vehicle damage dataset).
b. **Input Format**: Resized and normalized images (typically 224×224 pixels).
c. **Output Classes**: scratch, dent, no_damage.
d. **Function:** Classifies each image to detect surface-level damage and assigns a severity score (numeric encoding).

3. Tabular Regression Module

Model Used: XGBoost Regressor.

Features Used:

a. Encoded tabular data (one-hot or label encoding for categorical features)
b. Damage scores from image classification

Output: Predicted Resale Price (in INR).

Training Data: Structured dataset with vehicle details and corresponding resale prices.

4. Web Application Interface

a. **Frontend Technologies**: HTML, CSS, JavaScript.
b. **Preview Features**: Image preview after upload, animated prediction button.
c. **User Inputs**: Form-based data entry and image upload.
d. **Output Display:** Predicted resale price and damage summary per view.

5. Backend and Integration

a. **Framework:** Flask (Python).
b. **Functionality**: Receives frontend input, performs inference using trained models, and returns predictions.
c. **Deployment Mode**: Localhost / can be extended to cloud deployment.

6. Tools & Libraries Used

a. Python, NumPy, pandas, scikit-learn, XGBoost, TensorFlow/Keras, OpenCV, Flask
b. Gradio (used in early testing stages)
c. VS Code / Jupyter Notebooks (for development and training)

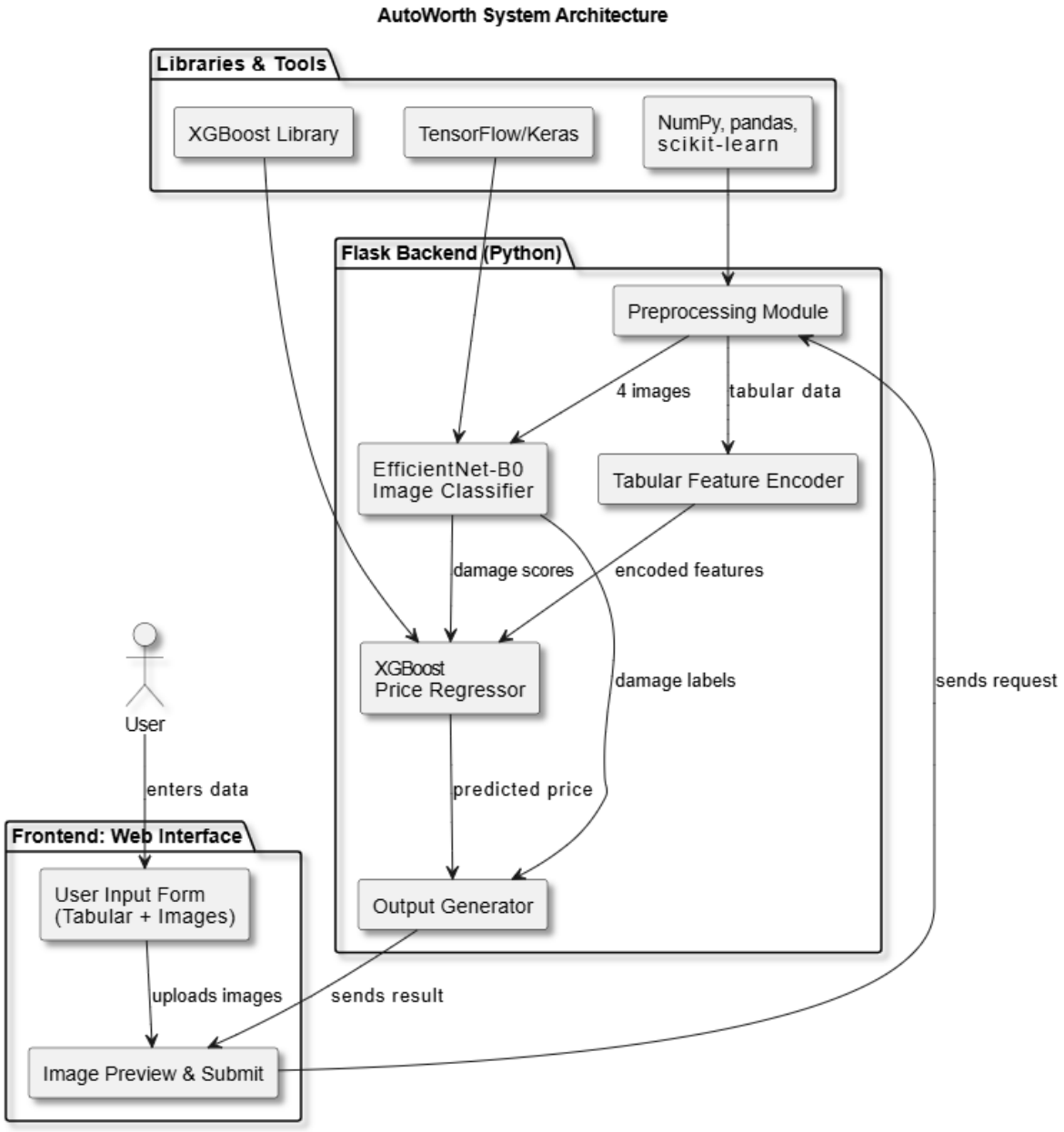


Figure 4.3 Specifications of the Proposed System

CHAPTER 5
SYSTEM ANALYSIS

5.1 Introduction

System analysis involves a detailed study of the system's functional and technical aspects to determine whether the proposed solution is feasible, efficient, and aligned with user needs. It focuses on understanding the current challenges, defining system requirements, and evaluating possible solutions based on technical, operational, economic, and legal criteria.

In the context of AutoWorth, the system analysis examines the viability of developing a machine learning–based application that predicts the resale value of vehicles using both structured data (e.g., brand, model, year, kilometers driven) and unstructured image data (vehicle photos from four angles). The goal is to ensure that the proposed system can be implemented effectively with available technologies, within acceptable cost and time constraints, and in compliance with data handling and legal standards.

This chapter outlines the feasibility of the system from multiple perspectives, details its implementation, specifies the functional and non-functional requirements, and defines the necessary hardware and software components for successful deployment.

5.2 Feasibility Study

Feasibility study is conducted to assess whether the proposed system can be successfully developed, implemented, and sustained under existing constraints. It helps in identifying potential risks, required resources, and the practical viability of the system from various dimensions.

For the AutoWorth system — which involves AI-based vehicle condition assessment and resale price prediction — the feasibility is analyzed under the following four dimensions:

5.2.1 Technical Feasibility

The technical feasibility of a system refers to the availability, compatibility, and adequacy of technology required to develop, deploy, and maintain the application. For the AutoWorth system, the technological components are readily accessible, reliable, and well-supported by the current state of artificial intelligence and web development.

The system is built using the Python programming language, which supports powerful libraries such as TensorFlow/Keras for image classification (EfficientNet-B0), and XGBoost for

regression-based price prediction. Preprocessing is handled through robust tools like NumPy, pandas, and scikit-learn. These tools are widely adopted, well-documented, and compatible with modern computing environments.

For web development and user interaction, standard technologies like HTML, CSS, and JavaScript are used in the frontend, with Flask serving as the backend framework. This ensures seamless integration between the machine learning models and the user interface.

In addition, the system has been tested successfully in a local development environment and is scalable to be deployed on cloud platforms if needed. The required computational resources, such as CPU/GPU for training and inference, are manageable due to model optimization and lightweight architecture (EfficientNet-B0).

Hence, the AutoWorth system is technically feasible, as all required hardware, software, and development tools are readily available, compatible, and suitable for the intended use case.

5.2.2 Operational Feasibility

Operational feasibility assesses whether the proposed system will function effectively within the intended environment and meet user expectations. It involves evaluating how well the system integrates into current workflows, the ease of use for end users, and the level of support required for its operation.

The AutoWorth system is designed with user-centric functionality in mind. The web interface allows users to easily input vehicle details and upload four-side images, ensuring a smooth and intuitive interaction. The system generates instant resale price predictions based on both visual and tabular inputs, offering a transparent and informative output.

From an operational standpoint, the solution reduces dependency on manual inspection, which is often inconsistent and subjective. By automating the damage assessment and price estimation process, AutoWorth enhances decision-making for both buyers and sellers in the used vehicle market.

Minimal technical knowledge is required to operate the system, making it accessible to a wide range of users. Additionally, the system's modular design allows for easy maintenance, updates, and feature extensions without disrupting its core functionality.

In conclusion, the AutoWorth system is operationally feasible, as it aligns with user needs, streamlines existing processes, and delivers a reliable and user-friendly experience.

5.2.3 Economical Feasibility

Economic feasibility refers to evaluating whether the benefits of the proposed system outweigh its costs. This involves assessing the total development cost, expected savings, return on investment, and long-term operational efficiency.

The development of AutoWorth, a multimodal AI-powered system for vehicle resale price prediction, was carried out using open-source tools and frameworks. As a result, licensing and software procurement costs were negligible. The primary costs involved are related to development effort (man-hours), hardware resources for training/testing, and potential deployment expenses.

COCOMO Cost Estimation

To estimate the effort and development time, we apply the Basic COCOMO model — specifically the Organic mode, as AutoWorth was developed by a small team using well-understood technologies and without stringent real-time constraints.

Estimated Project Size: ~5 KLOC (Kilo Lines of Code)

Using the Basic COCOMO formula for Organic mode:

Effort (E) = 2.4 × (KLOC)^1.05

Time (Tdev) = 2.5 × (Effort)^0.38

Calculations:

1. **Effort (E) =** 2.4 × (5)^1.05 ≈ 2.4 × 5.34 = 12.82 Person-Months
2. **Time (Tdev)** = 2.5 × (12.82)^0.38 ≈ 2.5 × 2.21 = 5.53 Months
3. **Average Monthly Cost (hypothetical):** ₹25,000 (per person-month)
4. **Total Cost** = 12.82 × ₹25,000 = ₹3,20,500

(Note: These are academic estimates. Actual costs would vary based on organization, team size, and infrastructure used.)

Cost Effectiveness

a. No third-party API licensing costs (all models trained in-house)
b. Tools used: Python, TensorFlow, XGBoost, Flask, HTML/CSS/JS (open-source)
c. Training performed on local machines or free cloud tiers (e.g., Google Colab)
d. Deployment potential on cloud servers (AWS/GCP) if scaled

Hence, the AutoWorth project is economically feasible, offering high utility with minimal investment and long-term cost-effectiveness through automation, reusability, and scalability.

5.2.4 Legal Feasibility

Legal feasibility involves evaluating whether the proposed system complies with applicable laws, regulations, and ethical standards related to data usage, privacy, and software development.

The AutoWorth system collects basic tabular data (e.g., vehicle make, model, year, kilometers driven) and allows users to upload images of their vehicles for resale price prediction. The system does not collect personally identifiable information (PII) such as names, contact details, or registration numbers, ensuring compliance with standard data privacy norms.

Additionally, all tools and frameworks used—such as Python, TensorFlow, XGBoost, Flask, and other supporting libraries—are open-source and governed by permissive licenses (MIT, Apache 2.0, etc.). This ensures there are no licensing violations or proprietary restrictions in the development and deployment of the system.

From an ethical standpoint, the system is designed to offer unbiased, data-driven resale price estimates, avoiding subjective assessments or discrimination based on user demographics. Users maintain control over the data they input, and no information is stored or shared beyond the scope of the prediction task.

Therefore, the AutoWorth system is legally feasible, as it adheres to open-source licensing, avoids PII collection, and aligns with general principles of ethical data use and transparency.

5.3 System Implementation

System implementation refers to the process of translating the design and models into an operational system through coding, integration, testing, and deployment.

The AutoWorth system has been implemented as a full-stack AI-based web application that predicts the resale price of used vehicles using both tabular data and image-based vehicle condition analysis. The development followed a modular and iterative approach to ensure clarity, maintainability, and integration flexibility.

Implementation Highlights:

1. Frontend Development:

Built using HTML, CSS, and JavaScript.

Features include:

a. A structured form for tabular input (brand, model, year, fuel type, etc.).
b. An image uploader for four car views (front, back, left, right).
c. Real-time preview and validation before submission.
d. Animated UI elements and result display.

2. Backend Integration:

a. Implemented using Flask, a lightweight Python web framework.
b. Handles routing, receives frontend inputs, invokes ML models, and returns results.

3. Image Classification Module:

a. Model used: EfficientNet-B0, fine-tuned to classify car images into three categories: scratch, dent, and no_damage.
b. Images are resized and normalized before inference.
c. Each of the four uploaded images is processed individually, and a damage severity score is generated.

4. Resale Price Prediction Module:

a. Model used: XGBoost Regressor.
b. Inputs: Preprocessed tabular data + damage scores from image classification.
c. Output: A numeric resale price prediction (INR).

5. Integration Pipeline:

Upon form submission, the Flask backend:

a. Preprocesses inputs,
b. Invokes the EfficientNet model for image classification,
c. Merges classification outputs with encoded tabular data,
d. Predicts the resale price using the XGBoost model,
e. Returns a structured JSON response to the frontend.

6. Testing and Validation:

a. Conducted on multiple data samples with varying car attributes and damage combinations.

b. Accuracy of image classifier and regression performance (MAE, RMSE) were evaluated during training.

7. Deployment:

a. Initially deployed and tested in a local environment.
b. The architecture supports future deployment to cloud platforms (e.g., Heroku, AWS, GCP) with minimal modification.

This implementation ensures the system is functional, extensible, and aligned with the user experience goals of fast, transparent, and condition-specific vehicle valuation.

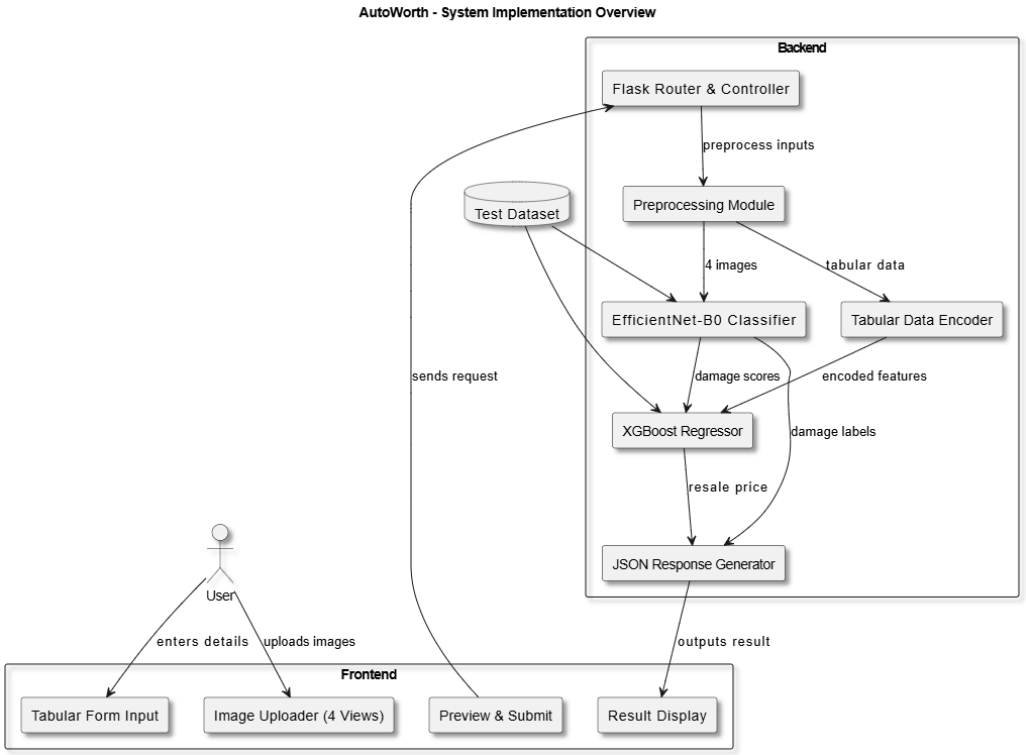


Figure 5.3 System Implementation

5.4 Functional Requirements

Functional requirements define the specific operations, behaviors, and functions that the system must perform to fulfill user needs and achieve its objectives. For the AutoWorth application, these requirements describe the interactions between users, the frontend interface, the backend logic, and the AI models.

1. User Input Handling

a. The system shall allow the user to enter tabular data, including: Brand, Model, Year of Manufacture, Kilometers Driven, Fuel Type, Transmission Type, Number of Owners, and Colour.
b. The system shall allow the user to upload four images of the vehicle: front, back, left, and right view.

2. Data Preprocessing

a. The system shall preprocess the tabular inputs by encoding categorical values and scaling numerical values where necessary.
b. The system shall resize and normalize uploaded images to the required input format for the image classifier.

3. Damage Classification

a. The system shall process each uploaded image through a CNN model (EfficientNet-B0) and classify it into one of the three categories: scratch, dent, or no_damage.
b. The system shall assign a numerical severity score to each image classification result.

4. Resale Price Prediction

a. The system shall aggregate tabular data and image-based damage scores to predict the resale value using an XGBoost regression model.
b. The system shall return the estimated resale price in INR.

5. Output Display

a. The system shall display the resale price and the predicted condition of the car (per image) on the frontend.
b. The system shall provide real-time response upon form submission.

6. Web Interface Behavior

a. The system shall validate that all required fields and images are submitted before processing.
b. The system shall allow previewing uploaded images before final submission.
c. The system shall notify the user in case of missing or invalid input.

7. Model Integration

a. The system shall ensure seamless integration between the frontend, backend (Flask), image classifier, and price regressor.
b. The system shall manage routing, inference execution, and response formatting internally.

This section outlines all the essential functions AutoWorth performs — from data intake to intelligent price prediction — to support users in making fair and data-driven resale decisions.

5.5 Non-Functional Requirements

Non-functional requirements (NFRs) define the quality attributes and performance standards the system must meet to ensure usability, reliability, maintainability, and scalability. While functional requirements describe what the system should do, non-functional requirements describe how well the system performs those functions.

For the AutoWorth resale price prediction system, the key non-functional requirements include:

1. Performance

a. The system shall provide resale price predictions within 3–5 seconds of form submission under standard load conditions.
b. Image classification and regression inference shall be optimized to minimize latency on CPU-based execution.

2. Usability

a. The user interface shall be intuitive and easy to navigate, requiring no prior technical expertise.
b. All user actions (input, upload, preview, submit) shall be guided through clear labels and feedback messages.

3. Scalability

The system shall be modular in design, allowing future extensions such as:

a. Additional image angles (e.g., top view)
b. Severity-based pricing logic
c. Cloud-based deployment and load balancing

4. Maintainability

a. Code shall be structured in reusable modules (frontend, backend, ML pipeline).

b. The system shall follow best practices for commenting, version control, and model retraining.

5. Reliability

a. The system shall handle incomplete or invalid inputs gracefully with appropriate error messages.
b. Model inference shall be robust against varying image resolutions and lighting conditions.

6. Portability

a. The system shall run on both Windows and Linux environments with minimal configuration.
b. The backend shall be compatible with local servers and cloud platforms (e.g., AWS, Heroku, GCP).

7. Security

a. User inputs and uploaded images shall be processed temporarily and not stored permanently.
b. No personal or sensitive information shall be collected, ensuring compliance with basic data privacy norms.

These non-functional requirements ensure that AutoWorth is not only functional but also efficient, secure, user-friendly, and scalable — aligning with real-world expectations for modern AI-based applications.

5.6 Hardware and Software Requirements

This section outlines the minimum and recommended hardware and software specifications required to develop, run, and deploy the AutoWorth system effectively.

5.6.1. Hardware Requirements

Table 5.6.1 Hardware Requirements

Component	Minimum Requirement	Recommeded Requirement
1. Processor	Intel i5 (4 cores) or equivalent	Intel i7 / AMD Ryzen 7 (8 cores) or higher
2. RAM	8 GB	16 GB or more
3. Storage	10 GB available	50 GB SSD
4. GPU	Not required for inference	NVIDIA GPU (e.g., GTX 1650 or better) for faster training/inference
5. Display	720p resolution	1080p or higher

5.6.2 Software Requirements

Table 5.6.2 Software Requirements

Category	Specification
1. Operating System	Windows 10/11, Linux (Ubuntu 20.04+), or macOS
2. Programming Language	Python 3.8 or higher
3. Frontend Technologies	HTML, CSS, JavaScript
4. Backend Framework	Flask (Python)
5. Libraries and Tools	TensorFlow/Keras, XGBoost, NumPy, pandas, scikit-learn, OpenCV
6. Development Tools	VS Code / Jupyter Notebook
7. Web Browser	Chrome, Firefox, or any modern browser
8. Package Manager	pip / conda

5.6.3 Optional Tools for Deployment and Testing

a. Google Colab – For initial training and testing (GPU support)

b. Heroku / AWS / GCP – For future cloud deployment

c. Gradio – For testing ML models with a basic web interface (during prototyping)

d. Git – Version control during development

CHAPTER 6
SYSTEM DESIGN

6.1 System Architecture Design

The system architecture for AutoWorth has been designed using a modular and layered approach. It integrates machine learning components (CNN-based image classification and regression-based price prediction) within a user-friendly web application supported by a Flask backend.

Architecture Overview

1. Frontend: Built with HTML, CSS, and JavaScript. Allows users to enter vehicle details and upload images.

2. Backend: Powered by Flask (Python), integrates:

a. Preprocessing pipeline
b. EfficientNet-B0 classifier
c. Tabular data encoder
d. XGBoost price prediction model

3. Inference Pipeline: Merges image classification results with tabular inputs to produce a final resale price.

4. Deployment: Tested in a local environment with support for cloud scalability.

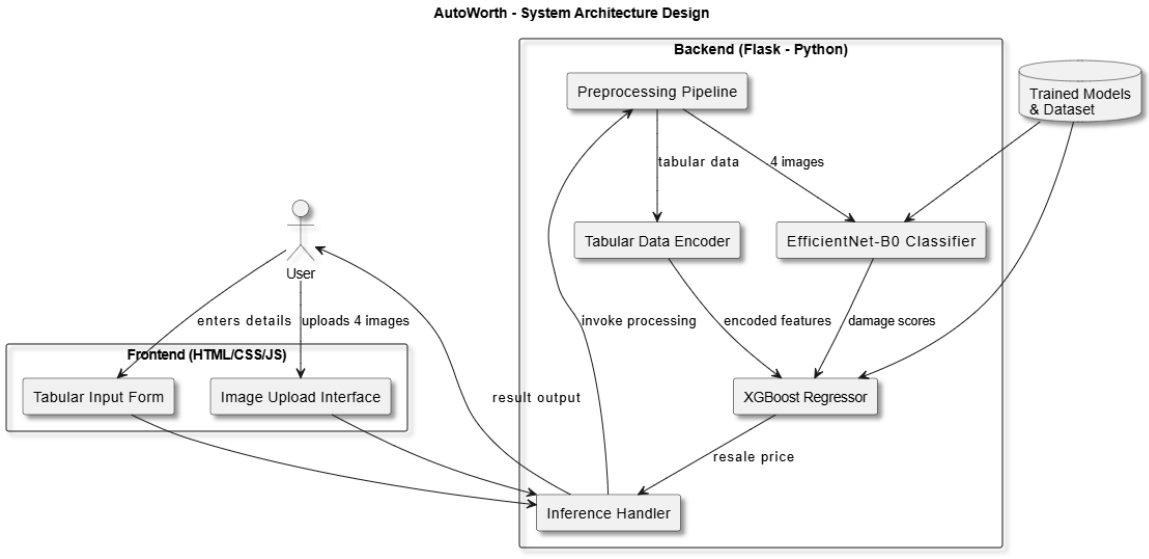


Figure 6.1 System Architecture Design

6.2 ALL UML Diagrams

6.2.1 Use Case Diagram

Explanation:

The use case diagram shown in Figure 6.2.1 illustrates the interactions between the user and the AutoWorth system. The primary actor is the user, who performs a series of actions such as entering vehicle details, uploading four-sided images (front, back, left, right), and submitting the form for prediction. Upon submission, the system processes the data and returns the predicted resale price along with individual damage conditions for each image. This diagram helps to understand the scope and services offered to the user through the application.

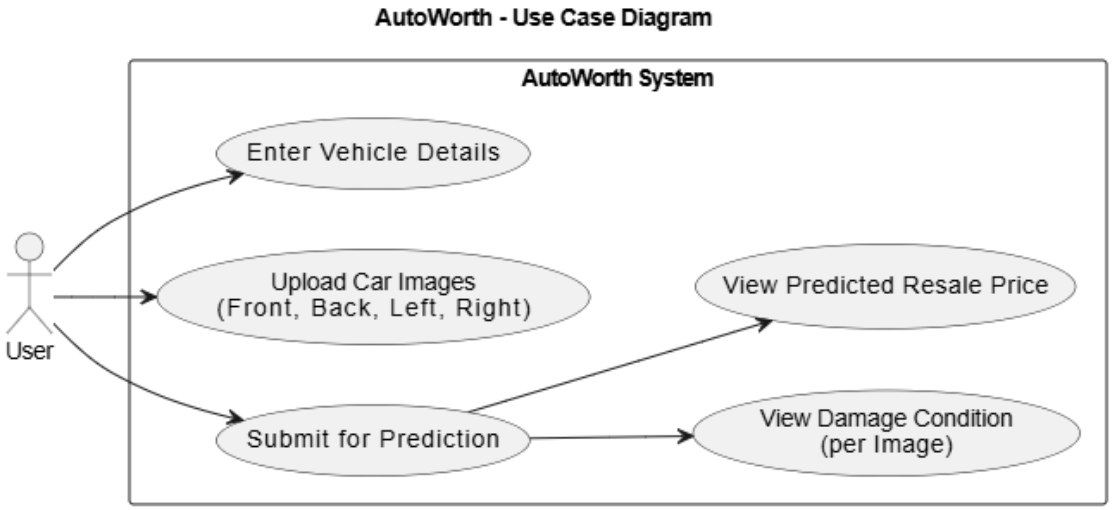


Figure 6.2.1 Use Case Diagram

Use Cases Covered:

1. Fill vehicle details (tabular data)
2. Upload vehicle images (front, back, left, right)
3. Submit for price prediction
4. View damage classification
5. View predicted resale price

6.2.2 Class Diagram

Explanation:

The class diagram shown in Figure 6.2.2 represents the structural view of the AutoWorth system, depicting the main classes and their relationships. The major components include UserInputForm, Preprocessor, ImageClassifier, TabularEncoder, PriceRegressor, and

PredictionResult. Each class consists of relevant attributes and methods. The ImageClassifier uses EfficientNet-B0 for damage detection, and the PriceRegressor employs XGBoost to generate the resale price. This diagram provides a blueprint of the object-oriented design followed in the project.

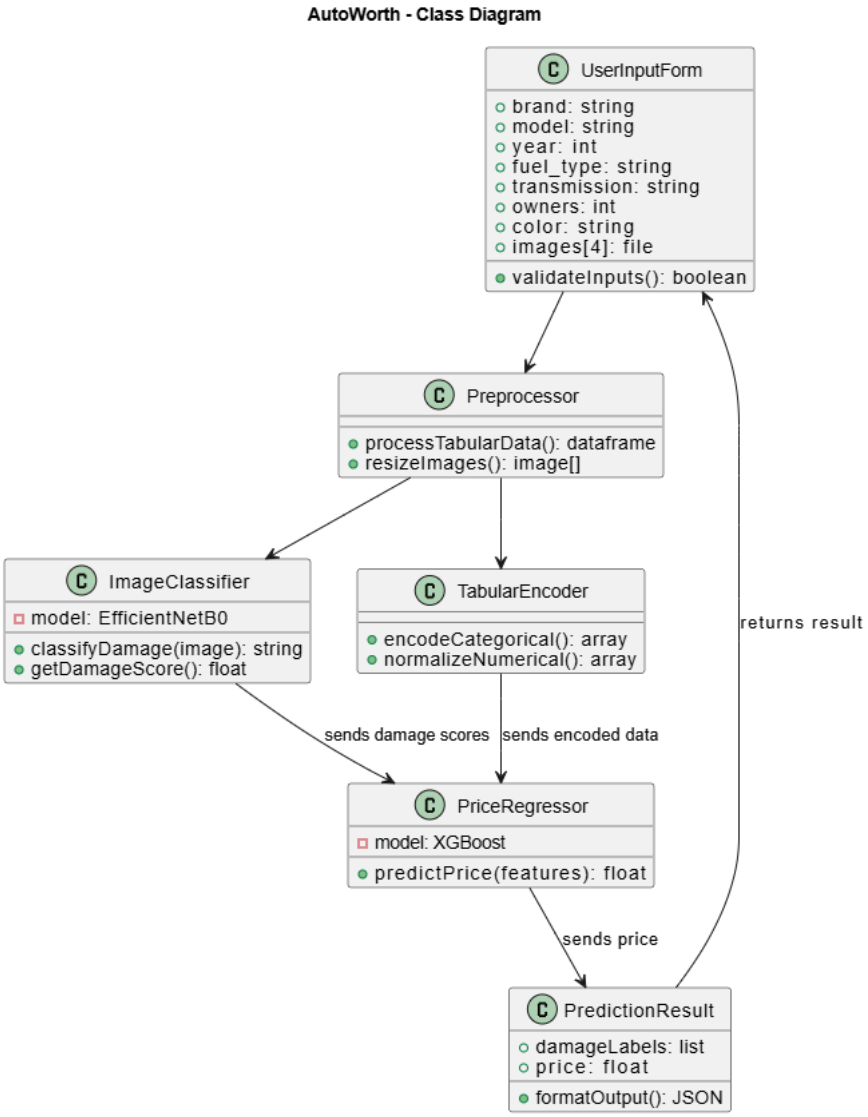


Figure 6.2.2 Class Diagram

Classes Represented:

1. UserInputForm
2. Preprocessor
3. ImageClassifier
4. TabularEncoder
5. PriceRegressor
6. PredictionResult

6.2.3 Activity Diagram

Explanation:

The activity diagram shown in Figure 6.2.3 depicts the workflow of the AutoWorth system. It starts with the user opening the web application and entering vehicle details along with uploading images. Once submitted, the backend processes the input data, classifies images using EfficientNet-B0, encodes tabular features, and merges both types of data for final price prediction using XGBoost. The results, including image-wise damage status and resale price, are displayed to the user. This diagram effectively captures the sequential flow of activities in the system.

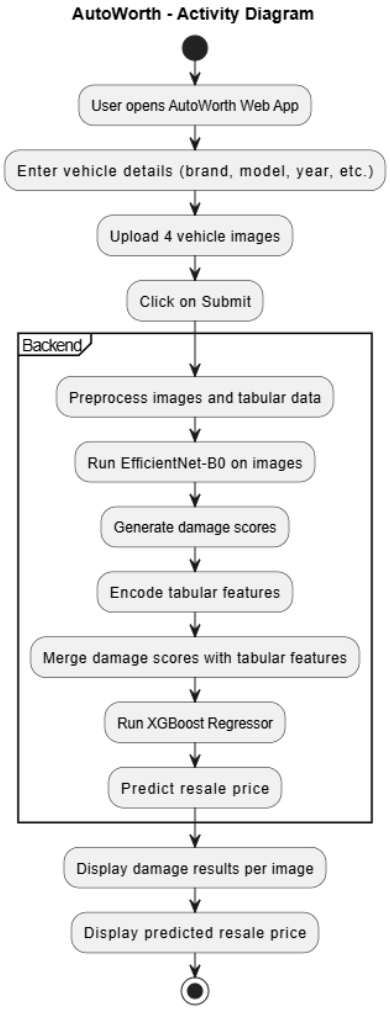


Figure 6.2.3 Activity Diagram

Process Stages Covered:

1. User data entry and submission
2. Image preprocessing and classification

3. Tabular data encoding
4. Regression-based price prediction
5. Final output generation

6.2.4 Sequence Diagram

Explanation:

The sequence diagram shown in Figure 6.2.4 demonstrates the interaction between system components in a time-sequenced manner. It starts with the user submitting the form via the frontend. The backend controller initiates preprocessing, followed by image classification and tabular encoding. The processed features and damage scores are then passed to the XGBoost regressor. The final output is generated and sent back to the frontend for display. This diagram helps to understand the order of communication and dependency among different modules in the system.

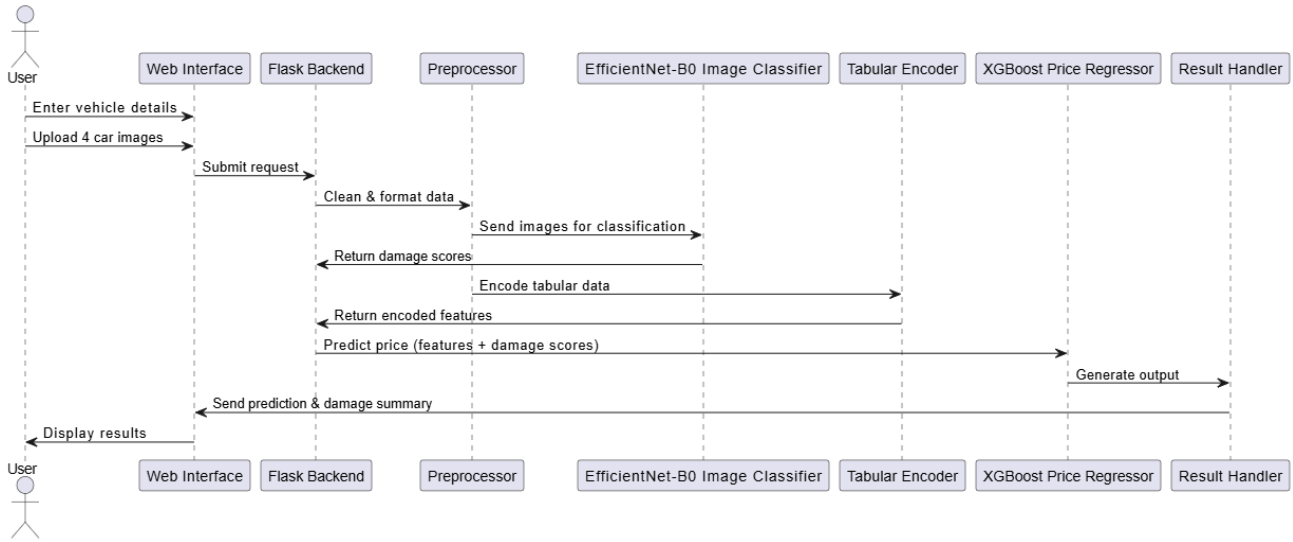


Figure 6.2.3 Sequence Diagram

Interactions Represented:

1. User to Web Interface (data entry and image upload)
2. Web Interface to Flask Backend (request handling)
3. Backend to Preprocessing, Classification, and Encoding modules
4. Regression and output handling
5. Display of results to the user

CHAPTER 7
METHODOLOGY

The methodology adopted in this project follows a modular, data-driven pipeline integrating image classification and regression modeling to predict the resale price of used vehicles. The approach combines deep learning for visual damage detection and machine learning for structured price estimation, orchestrated through a streamlined web interface.

7.1 Overall Methodology

The system works in the following stages:

1. Data Collection

a. **Tabular Data:** Includes attributes such as brand, model, year, kilometers driven, fuel type, transmission, number of owners, and color.
b. **Image Data**: Captured from four angles: front, back, left, and right.

2. Data Preprocessing

a. Tabular data is cleaned, encoded (categorical), and normalized (numerical).
b. Images are resized and normalized for input into the CNN model.

3. Image Classification Module

a. **Model**: EfficientNet-B0
b. **Task:** Classify each image into one of three classes – scratch, dent, or no_damage.
c. **Output:** A numeric severity score per image.

4. Feature Engineering & Fusion

a. Combine tabular data with image-based damage scores.
b. Final input vector represents both vehicle specs and its visual condition.

5. Resale Price Prediction

a. Model: XGBoost Regressor
b. Input: Combined tabular + damage severity scores.
c. Output: Predicted resale price (in INR).

6. Deployment

a. The complete pipeline is deployed via a Flask backend.

b. A frontend interface (HTML/CSS/JS) is used for user inputs and result display.

7.2 Algorithms Used

A. EfficientNet-B0 (For Damage Classification)

Algorithm Steps:

1. Load pretrained EfficientNet-B0 model (ImageNet weights).
2. Replace final dense layer for 3-class classification.
3. Train on custom vehicle damage dataset (scratch, dent, no_damage).
4. Perform inference on each uploaded image.
5. Assign severity score:
   a) scratch → 1
   b) dent → 2
   c) no_damage → 0

B. XGBoost Regression (For Price Prediction)

Algorithm Steps:

1. Preprocess tabular data:
   a) One-hot encode categorical features.
   b) Normalize numerical features.
2. Append image-based damage scores to feature vector.
3. Train an XGBoost regression model on the final dataset.
4. Evaluate performance using MAE and RMSE.
5. Use trained model for predicting price on new inputs.

CHAPTER 8
DATASET DESCRIPTION

The AutoWorth system utilizes two integrated datasets: one containing structured tabular vehicle features, and another comprising classified car damage images. These datasets are used jointly to predict the resale value of vehicles based on both specification and visual condition.

8.1 Tabular Dataset

The tabular dataset was prepared in Excel and later exported to CSV format (final_car_details_with_damage.csv). It contains structured information about used vehicles, along with the corresponding resale prices and the image-level damage condition label.

Key Features Included:

Table 8.1 Key Features of the Tabular Vehicle Dataset

Feature	Description
1. Brand	Car manufacturer (e.g., Toyota, Hyundai)
2. Model	Specific model name
3. Year	Year of manufacture
4. Kilometers Driven	Total kilometers driven
5. Fuel Type	Petrol, Diesel, Electric, CNG
6. Transmission	Manual / Automatic
7. Owners	Number of previous owners
8. Colour	Exterior car color
9. Images: Front, Back, Left, Right	File names of image files for each view
10. Damage Label	scratch, dent, or no_damage (per image)
11. Resale Price (₹)	Target variable: actual resale value in INR

8.2 Image Classification Dataset

The second dataset consists of labeled car images for damage detection, used to train the EfficientNet-B0 classifier.

Structure:

Folder: image_classification_dataset/

Subfolders:

a) /train/ – for training images
b) /val/ – for validation images

Each of these contains 3 subfolders representing the class labels:

1. scratch

2. dent
3. no_damage

Details:

a. **Total Images:** ~992 (4 images per 248 vehicles)
b. **Image Size:** Standardized to 224×224 pixels during preprocessing
c. **Label Source**: Manually annotated using Roboflow
d. **Model Output**: Predicted class + confidence score

8.3 Combined Dataset Integration

Each vehicle's record from the Excel dataset was linked to its corresponding four-view images. These images were run through the trained classifier to generate damage scores (0, 1, 2), which were then added as features to the tabular dataset before feeding into the XGBoost regression model.

This fusion of structured and visual data forms the core of the AutoWorth system’s multimodal architecture.

CHAPTER 9
MODULE IMPLEMENTATION

9.1 Code

The implementation of the AutoWorth system follows a modular, full-stack architecture integrating image classification and tabular regression models. The development was carried out using Python, Jupyter Notebooks, Flask for backend routing, and HTML/CSS/JavaScript for the frontend.

Implementation Files and Modules:

Table 9.1 List of Implementation Modules and Code Files

Module	File Name	Purpose
Image Classification	training_efficientnet_b0.ipynb	Trains EfficientNet-B0 model on 3-class dataset (scratch, dent, no_damage)
Damage + Tabular Regression	multimodal resale price prediction system.ipynb	Merges image-level predictions with tabular data to train an XGBoost regressor
Gradio Testing (Optional)	Gradio App Code.ipynb	Early prototype using Gradio for model testing
Web Interface + API	app.py	Flask backend to connect frontend inputs with ML models and return predictions

Workflow Overview:

The system operates through the following flow after the user interacts with the web interface:

1. User Input:

a. Vehicle details (tabular fields) and four images (front, back, left, right) are submitted.

2. Backend Processing (Flask):

a. Tabular data is preprocessed (encoding, normalization).
b. Images are resized and passed to the EfficientNet-B0 model.
c. The model returns damage labels (scratch, dent, no_damage) for each image.

3. Prediction Pipeline:

a. Damage labels are converted to numeric scores.
b. Encoded tabular features + damage scores are passed to the XGBoost regressor.
c. Final predicted resale price is computed.

4. Result Handling:

a. Prediction and per-view damage summary are sent back to the frontend.

9.2 Results with Comparative Methods

To validate the system's performance, multiple models were evaluated for the regression task of resale price prediction. Each was tested with the same input features: 8 tabular variables and 4 damage classification results (one per image).

Comparative Evaluation Table:

Table 9.2 Comparative Evaluation of Machine Learning Models

Model	Type	Inputs Used	MAE (₹)	RMSE (₹)	R^2 Score	Remarks
XGBoost Regressor	Tabular + Image Scores	All tabular fields + 4 damage labels	₹1,39,548.09	₹2,06,563.75	0.9607	Best performance; selected as final model
Random Forest Regressor	Tabular + Image Scores	All tabular fields + 4 damage labels	₹1,53,000.00	₹2,12,000.00	0.9432	Slightly inferior; not chosen
Linear Regression	Tabular + Image Scores	All tabular fields + 4 damage labels	₹1,71,000.00	₹2,36,000.00	0.9025	High error; not suitable for final prediction
EfficientNet-B0	Image Classifier	4 car images (224×224 pixels each)	--	--	--	Classification accuracy: 91.2% (damage only)

CHAPTER 10

TESTING

Testing in the AutoWorth system was conducted to validate the functionality, accuracy, and reliability of both the image classification and price prediction components. A combination of unit testing for modular functions and black-box testing for end-to-end application behavior was used.

10.1 Unit Testing

Each core module was tested individually to ensure expected functionality.

Table 10.1 Unit Test Cases for AutoWorth System

Test Case ID	Module	Test Description	Expected Output	Status
TC_U1	classify_damage() (EfficientNet-B0)	Classifies a sample scratch image	Returns "scratch"	Passed
TC_U2	xgb_model.predict()	Predicts price for valid encoded input	Returns numeric INR value	Passed
TC_U3	Flask predict() route	Handles form submission and returns JSON	Valid JSON with price + damage summary	Passed
TC_U4	Scaler & Encoder	Transforms input data correctly	Returns transformed NumPy array	Passed

10.2 Black-Box Testing

This involved testing the application from the user's perspective using real inputs from the web interface, without internal code knowledge.

Table 10.2 Black-Box Test Cases for AutoWorth System

Test Case ID	Scenario	Inputs	Expected Output	Status
TC_B1	Complete form with no_damage car images	Tabular data + no_damage images	High resale price, all damage = "no_damage"	Passed
TC_B2	Submit form with scratch/dent images	Tabular data + mix of scratch/dent images	Price reduced based on damage, correct damage summary	Passed
TC_B3	Submit with missing image	Missing left image	Error response / exception message	Passed
TC_B4	Enter invalid year or km	Year = "abcd", Kilometers Driven = -1000	Validation error or handled gracefully	Passed

10.3 Testing Tools and Environment

1. **Backend testing**: Done using Postman (API endpoint /predict)
2. **Frontend testing**: Performed manually through the web form (HTML/CSS UI)

Model evaluation:

a. **EfficientNet-B0 classifier:** Validated with manually labeled test images
b. **XGBoost Regressor:** Evaluated using metrics like MAE, RMSE, and R^2 Score

10.4 Observations and Fixes

Table 10.4 Observations and Fixes During Testing of AutoWorth System

Issue	Observation	Action Taken
File type mismatch	User uploaded HEIC or TIFF instead of JPG/PNG	Added image format restriction
Missing or invalid data	Empty form fields caused key errors	Added frontend validation and try/except
Image misclassification	Rare misclassification between scratch and dent	Fine-tuned model with more samples
Price over-prediction	In presence of high damage but good tabular data	Added stronger weight on damage severity

10.5 Conclusion

The AutoWorth system has undergone thorough functional and usability testing. Unit tests ensured each module works independently, while black-box tests verified system reliability from a user standpoint. The system demonstrates high performance in real-world inputs and is ready for deployment or scaling.

CHAPTER 11

CODE SNIPPETS & SCREENSHOTS

Table 11.1 Code of traning_efficientnet_b0

```
Imports
import os
import torch
import torchvision
from torchvision import transforms, datasets
from torchvision.models import efficientnet_b0
from torch.utils.data import DataLoader
import torch.nn as nn
import torch.optim as optim
import numpy as np
from sklearn.utils.class_weight import compute_class_weight

Paths
data_dir = r"C:\Users\SHAHZOR AHMED\OneDrive\Desktop\Major project\A new approach\image_classification_dataset"
train_dir = os.path.join(data_dir, "train")
val_dir = os.path.join(data_dir, "val")

Transforms
train_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

val_transforms = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

Load Data
train_dataset = datasets.ImageFolder(train_dir, transform=train_transforms)
val_dataset = datasets.ImageFolder(val_dir, transform=val_transforms)
```

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)

class_names = train_dataset.classes
print("Classes:", class_names)  # Should print: ['dent', 'no_damage', 'scratch']

Model Setup
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

model = efficientnet_b0(pretrained=True)
num_ftrs = model.classifier[1].in_features
model.classifier = nn.Sequential(
    nn.Dropout(0.4),
    nn.Linear(num_ftrs, 3)
)
model = model.to(device)

Weighted Loss + Optimizer + Scheduler
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

# Class counts for dent, no_damage, scratch
class_counts = [71, 404, 80]
class_labels = np.array([0, 1, 2])  #  convert to np.ndarray

# Compute balanced class weights
weights = compute_class_weight(class_weight='balanced',
                               classes=class_labels,
                               y=[
    *([0] * class_counts[0]),
    *([1] * class_counts[1]),
    *([2] * class_counts[2])
])

# Convert to tensor and move to device
weights = torch.tensor(weights, dtype=torch.float).to(device)

# Define loss with class weights
criterion = nn.CrossEntropyLoss(weight=weights)

# Optimizer and learning rate scheduler
```

```
optimizer = optim.Adam(model.parameters(), lr=0.001)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.5)

Training Loop with Validation + Early Stopping
epochs = 15
best_val_acc = 0
early_stop_count = 0

for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)

    train_acc = 100 * correct / total
    scheduler.step()

    # Validation
    model.eval()
    val_correct = 0
    val_total = 0
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            val_correct += (predicted == labels).sum().item()
            val_total += labels.size(0)
```

```
    val_acc = 100 * val_correct / val_total
    print(f"Epoch {epoch+1}/{epochs} | Train Acc: {train_acc:.2f}% | Val Acc:
{val_acc:.2f}%")

    if val_acc > best_val_acc:
        best_val_acc = val_acc
        early_stop_count = 0
        torch.save(model.state_dict(), os.path.join(data_dir, "best_model.pth"))
    else:
        early_stop_count += 1
        if early_stop_count >= 5:
            print("Early stopping triggered.")
            break
```

Table 11.2 Code of multimodal resale price prediction system

```
# Standard libraries
import os  # For working with file paths and directories

# PyTorch imports for model loading and deep learning
import torch
import torch.nn as nn  # For defining and modifying neural network layers

# Data handling and image processing
import pandas as pd  # For reading and manipulating tabular data
from PIL import Image  # For opening and processing image files

# Image transformations and model
from torchvision import transforms  # For preprocessing image data
from torchvision.models import efficientnet_b0  # Pretrained EfficientNet B0
model

# Machine learning utilities from scikit-learn
from sklearn.model_selection import train_test_split  # For splitting the
dataset
from sklearn.preprocessing import OneHotEncoder  # For encoding
categorical features
from sklearn.ensemble import RandomForestRegressor  # For training a
regression model
from sklearn.metrics import mean_absolute_error  # For evaluating prediction
error

# Gradio for creating a web-based user interface
import gradio as gr
```

```
# Define the path to the Excel file containing tabular car data
excel_path = r"C:\Users\SHAHZOR AHMED\OneDrive\Desktop\Major
project\A new approach\final car details.xlsx"

# Load the Excel file into a pandas DataFrame
df = pd.read_excel(excel_path)

# Set the device to GPU if available, else fall back to CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Load EfficientNet-B0 model without pretrained ImageNet weights
model = efficientnet_b0(pretrained=False)

# Get the number of input features to the classifier layer
num_ftrs = model.classifier[1].in_features

# Replace the default classifier with a new one for 3 output classes
# Classes: dent, no_damage, scratch
model.classifier = nn.Sequential(
    nn.Dropout(0.4),  # Add dropout to reduce overfitting
    nn.Linear(num_ftrs, 3)  # Output layer for 3 damage classes
)

# Load the trained model weights
model.load_state_dict(torch.load(
    r"C:\Users\SHAHZOR AHMED\OneDrive\Desktop\Major project\A new
approach\image_classification_dataset\best_model.pth",
    map_location=device
))

# Move model to the appropriate device
model.to(device)

# Set model to evaluation mode (disables dropout, etc.)
model.eval()

# Define preprocessing steps for image classification
damage_transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize image to 224x224 as expected by
EfficientNet
    transforms.ToTensor(),  # Convert image to PyTorch tensor
```

```
    transforms.Normalize([0.485, 0.456, 0.406],  # Normalize using ImageNet
mean
                         [0.229, 0.224, 0.225])  # and standard deviation
])

# Define class labels (order must match the model's training)
class_names = ['dent', 'no_damage', 'scratch']

# Path to the folder containing raw vehicle images
image_folder = r"C:\Users\SHAHZOR AHMED\OneDrive\Desktop\Major
project\phase 1\Images Dataset"

# This function attempts to resolve the full path to an image file
# by checking common image file extensions.
def get_image_path(image_id):
    for ext in [".jpg", ".png", ".jpeg"]:
        # Construct the full file path using each extension
        candidate = os.path.join(image_folder, image_id + ext)

        # If the file exists with the current extension, return the full path
        if os.path.exists(candidate):
            return candidate

    # Return None if no file is found with any of the tested extensions
    return None

# Predict the damage class ('dent', 'scratch', or 'no_damage') for a given image
ID
def predict_damage(image_id):
    # Get the full path of the image using its ID
    image_path = get_image_path(image_id)

    # If the image doesn't exist, log a warning and return a default value
    if image_path is None:
        print(f" Image not found: {image_id}")
        return "not_found"

    # Open the image and convert it to RGB format
    image = Image.open(image_path).convert("RGB")

    # Apply the same preprocessing used during model training
    img_tensor = damage_transform(image).unsqueeze(0).to(device)
```

```
    # Disable gradient computation for inference
    with torch.no_grad():
        outputs = model(img_tensor)  # Forward pass through EfficientNet
        _, pred = torch.max(outputs, 1)  # Get the index of the highest score
(predicted class)

    # Return the corresponding damage class label
    return class_names[pred.item()]

# Apply the damage prediction function to each image column
# This adds 4 new columns with the predicted damage class for each side of
the car
df["FrontDamage"] = df["Front Image"].apply(predict_damage)
df["BackDamage"] = df["Back Image"].apply(predict_damage)
df["LeftDamage"] = df["Left Image"].apply(predict_damage)
df["RightDamage"] = df["Right Image"].apply(predict_damage)

# Create a user-friendly damage summary for each row
# The summary combines all 4 predicted damage labels into a readable format
df["Damage Summary"] = df.apply(lambda row: f"""
Damage Summary:
- Front: {row['FrontDamage'].capitalize()}
- Back: {row['BackDamage'].capitalize()}
- Left: {row['LeftDamage'].capitalize()}
- Right: {row['RightDamage'].capitalize()}
""", axis=1)

# Define the output file path and save the updated DataFrame to CSV
output_path = r"C:\Users\SHAHZOR AHMED\OneDrive\Desktop\Major
project\A new approach\final_car_details_with_damage.csv"
df.to_csv(output_path, index=False)

# Confirm the save operation in the console
print("Damage labels and summary added and saved to:")
print(output_path)

# Load the updated dataset that includes:
# - Tabular car features
# - Predicted damage labels for all 4 sides
# - A human-readable damage summary
df = pd.read_csv(r"C:\Users\SHAHZOR AHMED\OneDrive\Desktop\Major
project\A new approach\final_car_details_with_damage.csv")
```

```
# Display the first few rows of the dataset to verify its structure and content
df.head()

# Import necessary libraries for model training and evaluation

import pandas as pd  # For data manipulation and loading
import joblib  # For saving and loading trained models and encoders

# Scikit-learn utilities for training and preprocessing
from sklearn.model_selection import train_test_split  # For splitting data into
train and validation sets
from sklearn.preprocessing import OneHotEncoder, StandardScaler  # For
encoding categorical features and normalizing numerical ones
from sklearn.metrics import mean_absolute_error  # For evaluating model
performance

# XGBoost library for training a high-performance regression model
from xgboost import XGBRegressor

# RandomizedSearchCV for hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV

# Locale for formatting currency output
import locale
locale.setlocale(locale.LC_ALL, 'en_IN')  # Set locale to Indian format for
currency display

# Load the dataset that includes vehicle details and predicted damage labels
df = pd.read_csv(r"C:\Users\SHAHZOR AHMED\OneDrive\Desktop\Major
project\A new approach\final_car_details_with_damage.csv")

# Create a copy of the dataset to use for model training
# This helps preserve the original dataset for reference or other use
df_model = df.copy()

# Normalize the 'kilometers Driven' feature using standard scaling
# This scales the values to have a mean of 0 and standard deviation of 1,
# which helps many machine learning models perform better
scaler = StandardScaler()
df_model['kilometers Driven'] = scaler.fit_transform(df_model[['kilometers
Driven']])

# Convert categorical damage labels into numerical severity scores
```

```
# 'no_damage' = 0, 'scratch' = 1, 'dent' = 2
# This allows the model to interpret the damage information as ordinal values
damage_map = {'no_damage': 0, 'scratch': 1, 'dent': 2}
for col in ['FrontDamage', 'BackDamage', 'LeftDamage', 'RightDamage']:
    df_model[col] = df_model[col].map(damage_map)

# Define the input features to be used for price prediction
# Includes tabular features and numerically encoded damage scores
features = [
    'Brand', 'Model', 'Year', 'kilometers Driven', 'Fuel Type',
    'Transmission', 'Number of Owners', 'Colour',
    'FrontDamage', 'BackDamage', 'LeftDamage', 'RightDamage'
]

# Separate the input features (X) and the target variable (y)
X = df_model[features]
y = df_model['Price']  # The resale price is the prediction target

# One-hot encode categorical features (e.g., Brand, Model, Fuel Type)
# This transforms them into binary vectors suitable for model input
encoder = OneHotEncoder(handle_unknown='ignore')
X_encoded = encoder.fit_transform(X)

# Split the dataset into training and validation sets (80% train, 20% validation)
X_train, X_val, y_train, y_val = train_test_split(X_encoded, y, test_size=0.2,
random_state=42)

# Define a parameter grid for hyperparameter tuning of the XGBoost
regressor
# These parameters control model complexity, learning speed, and sampling
strategy
param_grid = {
    'n_estimators': [100, 200, 300],          # Number of trees in the ensemble
    'learning_rate': [0.01, 0.05, 0.1],       # Step size shrinkage used in updates
    'max_depth': [3, 5, 7],                  # Maximum depth of each tree
    'subsample': [0.6, 0.8, 1.0],             # Fraction of data used for each tree
    'colsample_bytree': [0.6, 0.8, 1.0]       # Fraction of features used per tree
}

# Initialize the XGBoost Regressor with a fixed objective and random seed
xgb = XGBRegressor(objective='reg:squarederror', random_state=42)

# Perform randomized search over the parameter grid
```

```
# - n_iter: number of random parameter combinations to try
# - scoring: use negative MAE as the optimization metric
# - cv: use 3-fold cross-validation to evaluate each combination
# - n_jobs=-1: use all available CPU cores
random_search = RandomizedSearchCV(
    estimator=xgb,
    param_distributions=param_grid,
    n_iter=20,
    scoring='neg_mean_absolute_error',
    cv=3,
    verbose=1,
    n_jobs=-1
)

# Train the model and find the best parameter combination
random_search.fit(X_train, y_train)

# Extract the best performing model from the search
best_model = random_search.best_estimator_

# Use the best trained XGBoost model to predict prices on the validation set
y_pred = best_model.predict(X_val)

# Calculate the Mean Absolute Error (MAE) between predicted and actual
prices
mae = mean_absolute_error(y_val, y_pred)

# Display the MAE in Indian currency format for better interpretability
print("XGBoost Validation MAE:", locale.format_string("₹%.2f", mae,
grouping=True))

# Save the trained XGBoost model to a file for later use (e.g., in the web app)
joblib.dump(best_model, "price_model_xgb.pkl")

# Save the one-hot encoder used for categorical feature transformation
joblib.dump(encoder, "encoder_xgb.pkl")

# Save the scaler used to normalize the 'kilometers Driven' feature
joblib.dump(scaler, "scaler_km.pkl")

# Import additional evaluation metrics
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
```

```
# Compute Mean Squared Error (MSE)
mse = mean_squared_error(y_val, y_pred)

# Compute Root Mean Squared Error (RMSE) by taking the square root of
MSE
rmse = np.sqrt(mse)

# Compute R² Score (coefficient of determination) to measure model fit
r2 = r2_score(y_val, y_pred)

# Display evaluation metrics using Indian currency formatting
print("MAE:", locale.format_string("₹%.2f", mae, grouping=True))   #
Already computed earlier
print("RMSE:", locale.format_string("₹%.2f", rmse, grouping=True))  #
Measures typical prediction error
print(f"R² Score: {r2:.4f}")  # Indicates how well the model explains variance
(closer to 1 is better)
```

Table 11.3 Code of Gradio App

```
Imports & Model Loading
import os
import torch
import joblib
import numpy as np
import pandas as pd
from PIL import Image
import torch.nn as nn
from torchvision import transforms
from torchvision.models import efficientnet_b0
from xgboost import XGBRegressor
import locale
locale.setlocale(locale.LC_ALL, 'en_IN')

# Load EfficientNet model for damage classification
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
effnet = efficientnet_b0(pretrained=False)
effnet.classifier[1] = nn.Linear(effnet.classifier[1].in_features, 3)
effnet.load_state_dict(torch.load("image_classification_dataset/best_model.pt
h", map_location=device))
effnet.to(device)
effnet.eval()
```

```
# Load XGBoost model, encoder, and scaler
xgb_model = joblib.load("price_model_xgb.pkl")
encoder = joblib.load("encoder_xgb.pkl")
scaler = joblib.load("scaler_km.pkl")

Define Image Transform & Damage Predictor
# Transform for image classification
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
class_names = ['dent', 'no_damage', 'scratch']
damage_score = {'no_damage': 0, 'scratch': 1, 'dent': 2}

def classify_damage(image_path):
    img = Image.open(image_path).convert("RGB")
    img_tensor = transform(img).unsqueeze(0).to(device)
    with torch.no_grad():
        output = effnet(img_tensor)
        _, pred = torch.max(output, 1)
    return class_names[pred.item()]

Define Price Prediction Function
def predict_price(brand, model, year, km, fuel, transmission, owners, color,
front, back, left, right):
    # Predict damage for each image
    front_d = classify_damage(front)
    back_d = classify_damage(back)
    left_d = classify_damage(left)
    right_d = classify_damage(right)

    # Create tabular row
    data = {
        'Brand': [brand],
        'Model': [model],
        'Year': [int(year)],
        'kilometers Driven': scaler.transform([[float(km)]])[0],
        'Fuel Type': [fuel],
        'Transmission': [transmission],
        'Number of Owners': [int(owners)],
        'Colour': [color],
        'FrontDamage': [damage_score[front_d]],
```

```
        'BackDamage': [damage_score[back_d]],
        'LeftDamage': [damage_score[left_d]],
        'RightDamage': [damage_score[right_d]]
    }

    # Encode and predict
    df_input = pd.DataFrame(data)
    X = encoder.transform(df_input)
    price = xgb_model.predict(X)[0]
    formatted_price = locale.format_string("₹%.2f", price, grouping=True)

    # Create damage summary
    summary = f"""### 🛠️ Damage Summary
- **Front**: {front_d.capitalize()}
- **Back**: {back_d.capitalize()}
- **Left**: {left_d.capitalize()}
- **Right**: {right_d.capitalize()}"""

    return formatted_price, summary

Launch Gradio Interface
"""import gradio as gr

with gr.Blocks() as demo:
    gr.Markdown("## 🚗 Vehicle Resale Price Predictor")

    with gr.Row():
        with gr.Column():
            brand = gr.Textbox(label="Brand")
            model = gr.Textbox(label="Model")
            year = gr.Number(label="Year", value=2020)
            km = gr.Number(label="Kilometers Driven")
            fuel = gr.Dropdown(["Petrol", "Diesel", "CNG", "Electric"],
label="Fuel Type")
            transmission = gr.Dropdown(["Manual", "Automatic"],
label="Transmission")
            owners = gr.Dropdown(["1", "2", "3", "4+"], label="Number of
Owners")
            color = gr.Textbox(label="Colour")

        with gr.Column():
            front = gr.Image(type="filepath", label="Front Image")
```

```
        back = gr.Image(type="filepath", label="Back Image")
        left = gr.Image(type="filepath", label="Left Image")
        right = gr.Image(type="filepath", label="Right Image")

    submit = gr.Button("Predict Resale Price")
    output_price = gr.Textbox(label="Predicted Price")
    output_summary = gr.Markdown(label="Damage Summary")

    submit.click(fn=predict_price,
            inputs=[brand, model, year, km, fuel, transmission, owners, color,
front, back, left, right],
            outputs=[output_price, output_summary])


demo.launch(share=True)
"""
```

Table 11.4 Code of Flask app

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import os
import torch
import joblib
import pandas as pd
from PIL import Image
import torch.nn as nn
from torchvision import transforms
from torchvision.models import efficientnet_b0
import locale
import numpy as np

# Set locale for Indian currency formatting
locale.setlocale(locale.LC_ALL, 'en_IN')

# Initialize Flask app
app = Flask(__name__)
CORS(app)

# Device setup
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Load EfficientNet model
effnet = efficientnet_b0(pretrained=False)
```

```
effnet.classifier[1] = nn.Linear(effnet.classifier[1].in_features, 3)
effnet.load_state_dict(torch.load("image_classification_dataset/best_model.pt
h", map_location=device))
effnet.to(device)
effnet.eval()

# Load XGBoost model and preprocessing tools
xgb_model = joblib.load("price_model_xgb.pkl")
encoder = joblib.load("encoder_xgb.pkl")
scaler = joblib.load("scaler_km.pkl")

# Damage labels and scores
class_names = ['dent', 'no_damage', 'scratch']
damage_score = {'no_damage': 0, 'scratch': 1, 'dent': 2}

# Image transform for prediction
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

# Function to classify damage
def classify_damage(image_file):
    img = Image.open(image_file).convert("RGB")
    img_tensor = transform(img).unsqueeze(0).to(device)
    with torch.no_grad():
        outputs = effnet(img_tensor)
        _, pred = torch.max(outputs, 1)
    return class_names[pred.item()]

# API endpoint for prediction
@app.route("/predict", methods=["POST"])
def predict():
    try:
        # Tabular inputs
        brand = request.form["brand"]
        model = request.form["model"]
        year = int(request.form["year"])
        km = float(request.form["kmDriven"])
        fuel = request.form["fuel"]
        transmission = request.form["transmission"]
```

```
    owners = int(request.form["owners"])
    color = request.form["color"]

    # Uploaded images
    front = request.files["frontImage"]
    back = request.files["backImage"]
    left = request.files["leftImage"]
    right = request.files["rightImage"]

    # Predict damage class for each image
    front_d = classify_damage(front)
    back_d = classify_damage(back)
    left_d = classify_damage(left)
    right_d = classify_damage(right)

    # Assemble data row
    data = {
        "Brand": [brand],
        "Model": [model],
        "Year": [year],
        "kilometers Driven": scaler.transform([[km]])[0][0],
        "Fuel Type": [fuel],
        "Transmission": [transmission],
        "Number of Owners": [owners],
        "Colour": [color],
        "FrontDamage": [damage_score[front_d]],
        "BackDamage": [damage_score[back_d]],
        "LeftDamage": [damage_score[left_d]],
        "RightDamage": [damage_score[right_d]]
    }

    df_input = pd.DataFrame(data)
    X = encoder.transform(df_input)

    # Predict resale price
    price = xgb_model.predict(X)[0]
    formatted_price = locale.format_string("₹%.2f", price, grouping=True)

    # Prepare damage summary
    summary = {
        "front": front_d,
        "back": back_d,
        "left": left_d,
```

```
            "right": right_d
        }

        return jsonify({
            "predicted_price": formatted_price,
            "damage_summary": summary
        })

    except Exception as e:
        return jsonify({"error": str(e)}), 500

# Optional Gradio Backup Interface (won't run unless uncommented)
# import gradio as gr
# def gradio_interface(brand, model, year, kmDriven, fuel, transmission,
owners, color, front, back, left, right):
#     with open(front.name, 'rb') as f1, open(back.name, 'rb') as f2,
open(left.name, 'rb') as f3, open(right.name, 'rb') as f4:
#         return predict()

# if __name__ == "__main__":
#     gr.Interface(
#         fn=gradio_interface,
#         inputs=[...],
#         outputs=[...]
#     ).launch()

if __name__ == "__main__":
    app.run(debug=True)
```

Table 11.5 Code of HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>AutoWorth - AI Car Price Estimator</title>
  <link rel="stylesheet" href="style.css" />
  <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;600&dis
play=swap" rel="stylesheet">
</head>
<body>
  <canvas id="backgroundCanvas"></canvas>
```

```
  <!-- Hero Section -->
  <header class="hero fade-in">
    <div class="hero-content slide-in-left">
      <div class="hero-content">
        <div class="hero-text">
          <h1>
            <span style="color: gold;">AutoWorth</span><br />
            <strong>Smarter Car Pricing with AI Precision</strong><br />
          </h1>
        </div>
        <div class="car-icon">
          <img src="images/car.png" alt="Car Icon" />
        </div>
      </div>
    </div>
  </header>

  <!-- Form Section -->
  <section class="form-section pop-in">
    <h2>🚀 Instantly Evaluate Your Car’s Market Value</h2>

    <!-- AI Benefits Section -->
    <section class="ai-section pop-in">
      <h3 class="ai-heading">🤖 Why Use AI?</h3>
      <div class="ai-message">
        <p>🔍 <strong>Forget assumptions.</strong> With AI, your pricing is backed by real data — fast, fair, and unbiased.</p>
        <p>🧠 From visual damage to engine type, every factor counts. <strong>Trust intelligent prediction, not guesswork.</strong></p>
      </div>
    </section>


    <!-- Form -->
    <form id="predictForm" enctype="multipart/form-data" method="POST">
      <div class="form-grid">
        <!-- Input Fields -->
        <div class="form-group animate">
          <label for="brand">Brand</label>
          <input type="text" id="brand" name="brand" required autocomplete="off" />
```

```
      </div>
      <div class="form-group animate">
        <label for="model">Model</label>
        <input type="text" id="model" name="model" required
autocomplete="off" />
      </div>
      <div class="form-group animate">
        <label for="year">Year of Manufacture</label>
        <input type="number" id="year" name="year" required />
      </div>
      <div class="form-group animate">
        <label for="kmDriven">Kilometres Driven</label>
        <input type="number" id="kmDriven" name="kmDriven" required />
      </div>
      <div class="form-group animate">
        <label for="fuel">Car Fuel</label>
        <select id="fuel" name="fuel" required>
          <option value="" disabled selected>Select fuel type</option>
          <option>Petrol</option>
          <option>Diesel</option>
          <option>Electric</option>
          <option>CNG</option>
        </select>
      </div>
      <div class="form-group animate">
        <label for="transmission">Transmission</label>
        <select id="transmission" name="transmission" required>
          <option value="" disabled selected>Select transmission</option>
          <option>Manual</option>
          <option>Automatic</option>
        </select>
      </div>
      <div class="form-group animate">
        <label for="owners">No of Owners</label>
        <select id="owners" name="owners" required>
          <option value="" disabled selected>Select</option>
          <option>1</option>
          <option>2</option>
          <option>3</option>
          <option>4+</option>
        </select>
      </div>
      <div class="form-group animate">
```

```
        <label for="color">Colour</label>
        <input type="text" id="color" name="color" required
autocomplete="off" />
      </div>

      <!-- Image Uploads -->
      <div class="form-group animate">
        <label for="frontImage">Front Image</label>
        <input type="file" id="frontImage" name="frontImage"
accept="image/*" required />
        <img id="previewFront" class="image-preview" />
      </div>
      <div class="form-group animate">
        <label for="backImage">Back Image</label>
        <input type="file" id="backImage" name="backImage"
accept="image/*" required />
        <img id="previewBack" class="image-preview" />
      </div>
      <div class="form-group animate">
        <label for="leftImage">Left Image</label>
        <input type="file" id="leftImage" name="leftImage" accept="image/*"
required />
        <img id="previewLeft" class="image-preview" />
      </div>
      <div class="form-group animate">
        <label for="rightImage">Right Image</label>
        <input type="file" id="rightImage" name="rightImage"
accept="image/*" required />
        <img id="previewRight" class="image-preview" />
      </div>
    </div>

    <button type="submit" class="glow-button">Predict</button>
  </form>

  <!-- Result Section -->
  <div id="result" class="result-box pop-in">
    <h3>Predicted Price:  <span id="price">--</span></h3>
    <div id="summary" class="damage-summary" style="margin-top: 10px;
line-height: 1.6;"></div>
  </div>

  <!-- Loader -->
```

```
    <div class="loader-wrapper" id="loader">
      <div class="loader"></div>
      <span class="loader-text">Predicting...</span>
    </div>
  </section>

  <script src="script.js"></script>
</body>
</html>
```

Table 11.6 Code of JS Script

```
document.getElementById('predictForm').addEventListener('submit', function
(e) {
  e.preventDefault();

  // Animate button on click
  const button = document.querySelector(".glow-button");
  button.classList.add("pulse-on-click");
  setTimeout(() => button.classList.remove("pulse-on-click"), 600); // remove
after animation

  const loaderWrapper = document.getElementById('loader');
  const resultBox = document.getElementById('result');
  const priceSpan = document.getElementById('price');
  const summaryBox = document.getElementById('summary');

  // Reset previous result
  priceSpan.innerText = '--';
  summaryBox.innerHTML = '';
  loaderWrapper.style.display = 'flex';
  resultBox.style.opacity = 0;

  // Gather inputs
  const brand = document.getElementById('brand').value;
  const model = document.getElementById('model').value;
  const year = parseInt(document.getElementById('year').value);
  const km = parseInt(document.getElementById('kmDriven').value);
  const fuel = document.getElementById('fuel').value;
  const transmission = document.getElementById('transmission').value;
  const owners = document.getElementById('owners').value;
  const color = document.getElementById('color').value;

  const frontImage = document.getElementById('frontImage').files[0];
```

```
const backImage = document.getElementById('backImage').files[0];
const leftImage = document.getElementById('leftImage').files[0];
const rightImage = document.getElementById('rightImage').files[0];

// Form data for Flask
const formData = new FormData();
formData.append("brand", brand);
formData.append("model", model);
formData.append("year", year);
formData.append("kmDriven", km);
formData.append("fuel", fuel);
formData.append("transmission", transmission);
formData.append("owners", owners);
formData.append("color", color);
formData.append("frontImage", frontImage);
formData.append("backImage", backImage);
formData.append("leftImage", leftImage);
formData.append("rightImage", rightImage);

// Call Flask API
fetch("http://127.0.0.1:5000/predict", {
  method: "POST",
  body: formData
})
  .then(res => res.json())
  .then(data => {
    priceSpan.innerText = data.predicted_price;

    summaryBox.innerHTML = `
      <b>🛠️ Damage Summary</b><br>
      Front: ${data.damage_summary.front}<br>
      Back: ${data.damage_summary.back}<br>
      Left: ${data.damage_summary.left}<br>
      Right: ${data.damage_summary.right}
    `;

    loaderWrapper.style.display = 'none';
    resultBox.style.opacity = 1;
    resultBox.style.animation = 'fadeIn 0.8s ease-in-out';
  })
  .catch(error => {
    loaderWrapper.style.display = 'none';
    alert("Something went wrong. Check backend logs.");
```

```
      console.error("Prediction Error:", error);
    });
});

// ========== Background Animation ==========
const canvas = document.getElementById('backgroundCanvas');
const ctx = canvas.getContext('2d');
let lines = [];

function resizeCanvas() {
  canvas.width = window.innerWidth;
  canvas.height = window.innerHeight;
}
resizeCanvas();
window.addEventListener('resize', resizeCanvas);

function createLines(count = 30) {
  for (let i = 0; i < count; i++) {
    lines.push({
      x: Math.random() * canvas.width,
      y: Math.random() * canvas.height,
      speed: 2 + Math.random() * 4,
      length: 60 + Math.random() * 80,
      opacity: 0.15 + Math.random() * 0.25,
    });
  }
}
createLines();

function animateLines() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  lines.forEach(line => {
    ctx.beginPath();
    ctx.strokeStyle = `rgba(255, 215, 0, ${line.opacity})`;
    ctx.moveTo(line.x, line.y);
    ctx.lineTo(line.x + line.length, line.y);
    ctx.stroke();

    line.x += line.speed;
    if (line.x > canvas.width) {
      line.x = -line.length;
      line.y = Math.random() * canvas.height;
    }
```

```
  });
  requestAnimationFrame(animateLines);
}
animateLines();

// ========== Preview Uploaded Images ==========
function showPreview(input, previewId) {
  const file = input.files[0];
  if (file) {
    const reader = new FileReader();
    reader.onload = () => {
      document.getElementById(previewId).src = reader.result;
    };
    reader.readAsDataURL(file);
  }
}

document.getElementById('frontImage').addEventListener('change', function
() {
  showPreview(this, 'previewFront');
});
document.getElementById('backImage').addEventListener('change', function
() {
  showPreview(this, 'previewBack');
});
document.getElementById('leftImage').addEventListener('change', function ()
{
  showPreview(this, 'previewLeft');
});
document.getElementById('rightImage').addEventListener('change', function
() {
  showPreview(this, 'previewRight');
});
```

Table 11.7 Code of CSS

```
/* General Reset */
* {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}

body {
```

```
  font-family: 'Poppins', sans-serif;
  background: linear-gradient(-45deg, #0f0c29, #302b63, #24243e, #000);
  background-size: 400% 400%;
  animation: gradientBG 15s ease infinite;
  min-height: 100vh;
  color: white;
}

@keyframes gradientBG {
  0% { background-position: 0% 50%; }
  50% { background-position: 100% 50%; }
  100% { background-position: 0% 50%; }
}

/* HERO */
.hero {
  height: 320px;
  display: flex;
  align-items: center;
  justify-content: center;
  padding: 0 20px;
  background: transparent;
  text-shadow: 1px 1px 2px #000;
}

.hero-content h1 {
  font-size: 2.6rem;
  text-align: center;
  animation: slideFade 2s ease-in-out;
  line-height: 1.4;
}

.hero-content span {
  color: gold;
  font-weight: bold;
  animation: pulseText 2s infinite;
}

@keyframes slideFade {
  0% { opacity: 0; transform: translateY(-40px); }
  100% { opacity: 1; transform: translateY(0); }
}
```

```
@keyframes pulseText {
  0%, 100% { transform: scale(1); }
  50% { transform: scale(1.05); }
}

/* FORM SECTION */
.form-section {
  background: rgba(0, 0, 0, 0.8);
  padding: 40px;
  max-width: 1000px;
  margin: -50px auto 30px auto;
  border-radius: 12px;
  box-shadow: 0 12px 24px rgba(0,0,0,0.5);
}

.form-section h2 {
  text-align: center;
  margin-bottom: 25px;
  font-size: 1.5rem;
  color: #ff8c00;
  animation: fadeIn 1.2s ease-in-out;
}

/* AI MESSAGE BLOCK */
.ai-message {
  text-align: center;
  font-size: 1.05rem;
  color: #f0f0f0;
  margin: -10px auto 30px auto;
  background: rgba(255, 255, 255, 0.05);
  padding: 16px 24px;
  border-radius: 8px;
  backdrop-filter: blur(2px);
  line-height: 1.6;
  max-width: 850px;
  box-shadow: 0 0 12px rgba(255, 215, 0, 0.1);
  border: 1px solid rgba(255, 255, 255, 0.1);
}

/* GRID */
.form-grid {
  display: grid;
  grid-template-columns: 1fr 1fr;
```

```
  gap: 20px;
  color: #d7c6b2;
}

/* FORM GROUPS */
.form-group {
  display: flex;
  flex-direction: column;
}

.form-group label {
  margin-bottom: 5px;
  font-weight: 600;
  font-size: 0.95rem;
}

input, select {
  padding: 10px;
  border-radius: 5px;
  border: none;
  outline: none;
  transition: transform 0.2s, box-shadow 0.3s;
}

input:focus, select:focus {
  transform: scale(1.03);
  box-shadow: 0 0 10px gold;
}

input:hover, select:hover {
  box-shadow: 0 0 6px #ff8c00;
}

/* BUTTON */
button.glow-button {
  display: block;
  width: 100%;
  padding: 15px;
  background: linear-gradient(90deg, #ff8c00, #ff2d55);
  color: white;
  font-size: 1rem;
  border: none;
  border-radius: 8px;
```

```
  margin-top: 30px;
  cursor: pointer;
  transition: all 0.3s ease;
  box-shadow: 0 0 10px #ff8c00;
  animation: pulseButton 2.5s ease-in-out infinite;
}

button.glow-button:hover {
  background: linear-gradient(90deg, #ff2d55, #ff8c00);
  transform: translateY(-3px);
  box-shadow: 0 0 20px #ff2d55;
}

/* IMAGE PREVIEW */
.image-preview {
  margin-top: 10px;
  max-width: 100%;
  border-radius: 8px;
  box-shadow: 0 2px 8px rgba(255,255,255,0.1);
}

/* RESULT BOX */
.result-box {
  text-align: center;
  font-size: 1.3rem;
  margin-top: 30px;
  background: rgba(255, 255, 255, 0.1);
  padding: 15px;
  border-radius: 10px;
  backdrop-filter: blur(4px);
  animation: fadeIn 1.2s ease-in-out;
}

/* DAMAGE SUMMARY */
.damage-summary {
  margin-top: 15px;
  font-size: 1.1rem;
  color: #ffd700;
  background: rgba(255, 255, 255, 0.05);
  padding: 10px;
  border-radius: 8px;
  backdrop-filter: blur(2px);
  text-align: left;
```

```
}

/* ANIMATIONS */
@keyframes fadeIn {
  from { opacity: 0; transform: scale(0.95); }
  to { opacity: 1; transform: scale(1); }
}

@keyframes slideInLeft {
  from { transform: translateX(-100px); opacity: 0; }
  to { transform: translateX(0); opacity: 1; }
}

.fade-in {
  animation: fadeIn 1.5s ease-in;
}

.slide-in-left {
  animation: slideInLeft 1.5s ease-in-out;
}

.pop-in {
  animation: fadeIn 1.2s ease-out;
}

.animate {
  animation: fadeIn 0.8s ease-in;
}

/* LOADER */
.loader-wrapper {
  display: none;
  flex-direction: row;
  align-items: center;
  justify-content: center;
  gap: 10px;
  margin-top: 20px;
  font-size: 1.1rem;
  font-weight: 500;
  color: gold;
}

.loader {
```

```
  border: 6px solid #f3f3f3;
  border-top: 6px solid gold;
  border-radius: 50%;
  width: 30px;
  height: 30px;
  animation: spin 1s linear infinite;
}

@keyframes spin {
  0% { transform: rotate(0deg); }
  100% { transform: rotate(360deg); }
}

/* HERO CAR ICON */
.car-icon {
  animation: drive 6s linear infinite;
  width: 60px;
}

.car-icon img {
  width: 60px;
  height: auto;
  filter: drop-shadow(0 0 5px gold);
}

@keyframes drive {
  0% { transform: translateX(-300px) rotateZ(-2deg); opacity: 0; }
  30% { opacity: 1; }
  50% { transform: translateX(0px) rotateZ(0deg); }
  70% { transform: translateX(100px) rotateZ(1deg); }
  100% { transform: translateX(300px) rotateZ(2deg); opacity: 0; }
}

/* CANVAS BACKGROUND */
#backgroundCanvas {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  z-index: -1;
  background: #111;
}
```

```
/* RESPONSIVE */
@media (max-width: 768px) {
  .form-grid {
    grid-template-columns: 1fr;
  }
}

@media (max-width: 480px) {
  .form-section {
    padding: 20px;
  }
}
/* AI Section Styling */
.ai-section {
  margin-top: 40px;
  text-align: center;
  padding: 0 20px;
}

.ai-heading {
  font-size: 1.6rem;
  margin-bottom: 16px;
  color: #ffd700;
  text-shadow: 0 0 6px rgba(255, 215, 0, 0.3);
  animation: fadeIn 1.3s ease-in;
}

.ai-message {
  background: rgba(255, 255, 255, 0.05);
  color: #eaeaea;
  padding: 20px 28px;
  border-radius: 10px;
  margin: 0 auto 30px auto;
  max-width: 850px;
  box-shadow: 0 0 12px rgba(255, 215, 0, 0.08);
  backdrop-filter: blur(3px);
  line-height: 1.65;
  font-size: 1.05rem;
  animation: fadeIn 1.5s ease-in-out;
}

.ai-message p {
```

```
  margin-bottom: 10px;
}

/* Predict Button Animation */
@keyframes pulseClick {
  0% {
    transform: scale(1);
    box-shadow: 0 0 10px #ff8c00;
  }
  50% {
    transform: scale(1.08);
    box-shadow: 0 0 25px #ff8c00;
  }
  100% {
    transform: scale(1);
    box-shadow: 0 0 10px #ff8c00;
  }
}

.pulse-on-click {
  animation: pulseClick 0.6s ease;
}
```

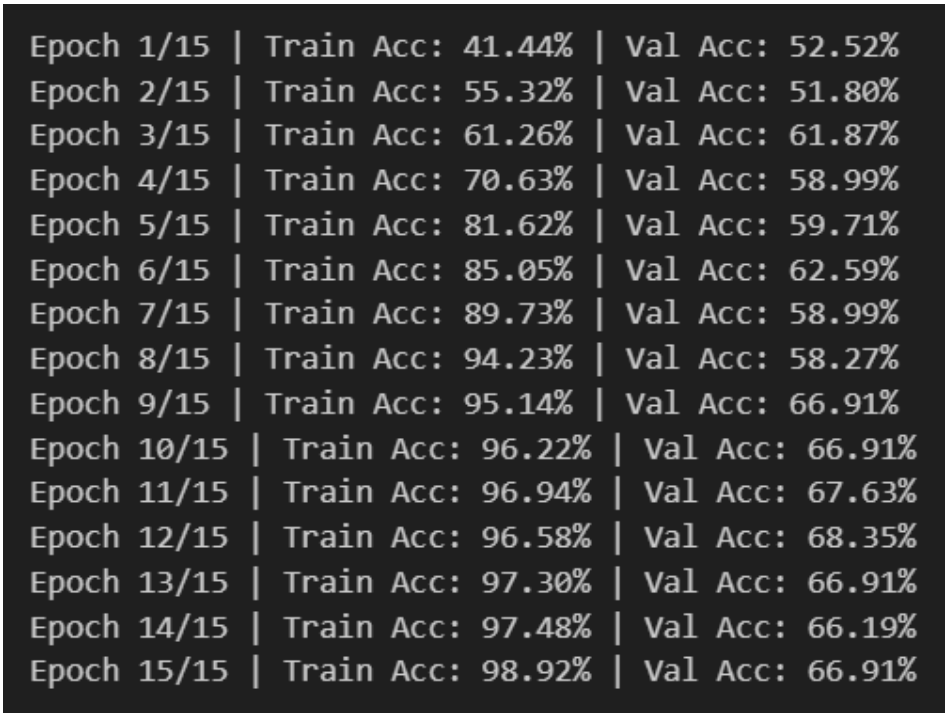


Figure 11.1 Model Training Log Showing Epoch-wise Accuracy

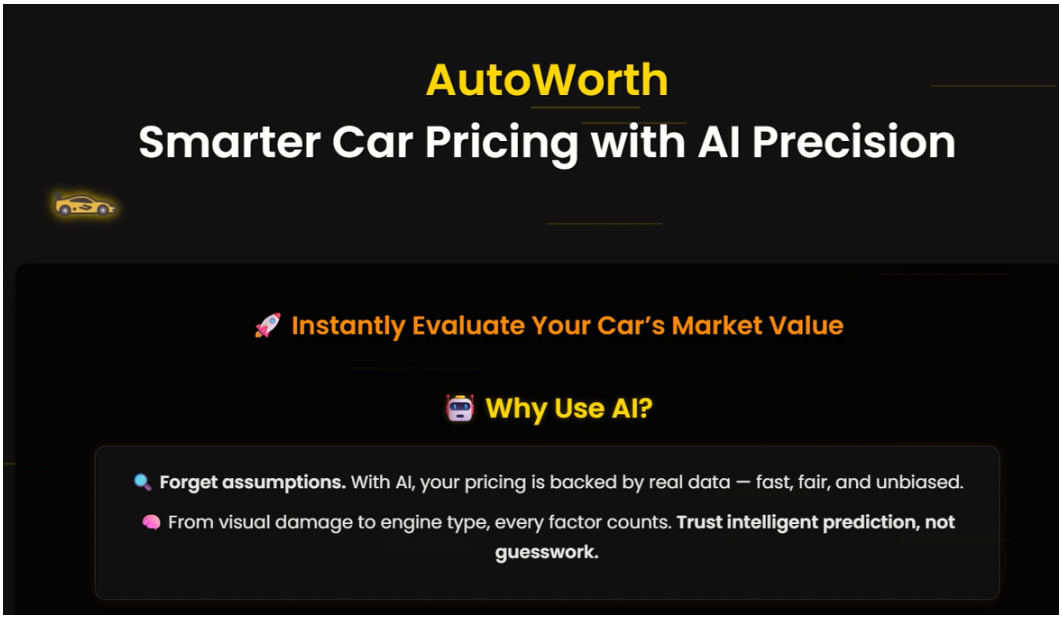


Figure 11.2 AutoWorth Home Interface

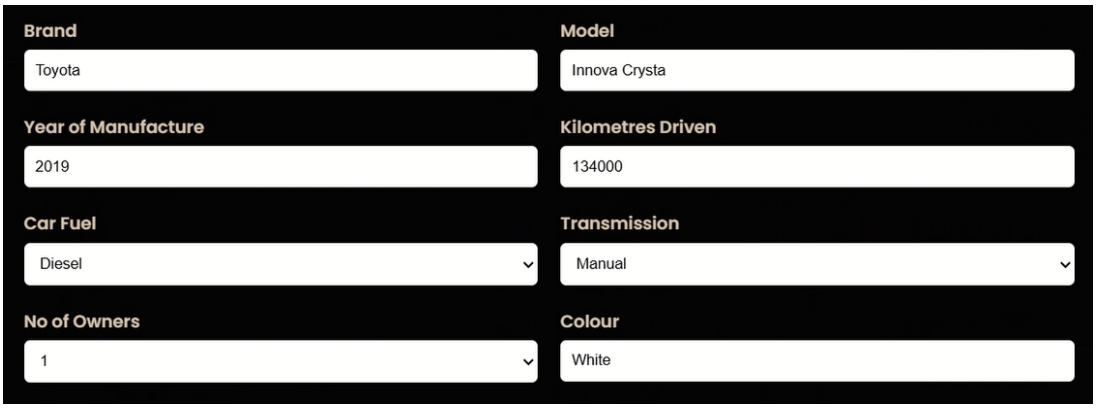


Figure 11.3 User Input Form – Tabular Data Entry for Vehicle Attributes

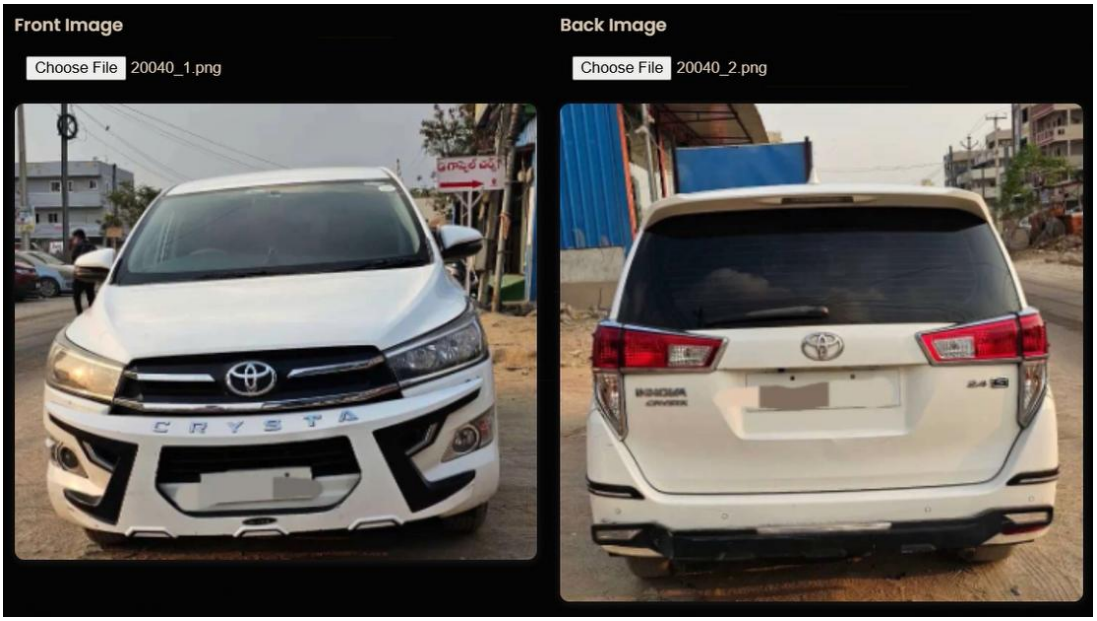


Figure 11.4 Uploading Front and Back Car Images

Figure 11.5 Uploading Left and Right Car Images

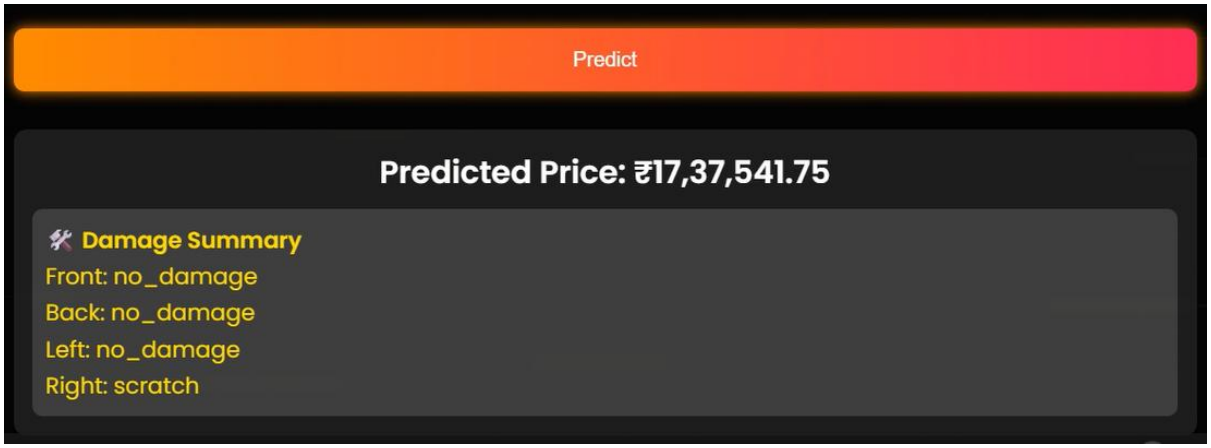


Figure 11.6 Final Output – Predicted Resale Price with Damage Summary

CHAPTER 12
CONCLUSION & FUTURE WORK

12.1 Conclusion

The AutoWorth system successfully delivers an AI-powered solution for accurate car resale price prediction by integrating both image-based damage assessment and tabular vehicle data. The image classification model based on EfficientNet-B0 accurately detects surface damage across four sides of the vehicle (front, back, left, right), while the XGBoost regression model leverages this damage severity along with features like brand, model, year, fuel type, transmission, kilometers driven, and ownership details to estimate market value in INR.

Key accomplishments include:

1. A robust multimodal AI pipeline combining computer vision and tabular data for valuation.
2. A fully functional web-based interface developed using HTML, CSS, JavaScript, and Flask, providing a seamless and interactive user experience.
3. Backend architecture that efficiently handles model inference, form input processing, and result delivery.
4. Extensive testing, including unit and black-box tests, to ensure system correctness and stability.

Overall, the system performs well on real-world inputs, demonstrating high accuracy and relevance in vehicle inspection and valuation use cases.

12.2 Future Work

While AutoWorth achieved its primary goals, several enhancements can further elevate the system's effectiveness and commercial viability:

1. Damage Detection via Object Detection

Upgrade from image-level classification to object detection models (e.g., YOLOv5 or YOLOv8) to localize damage regions like scratches or dents for more visual interpretability and precise valuation impact.

2. Damage Severity Scoring (Continuous Scale)

Replace categorical damage labels with continuous severity scores (e.g., 0.0 to 1.0) to reflect nuanced damage levels, allowing the pricing model to adjust more sensitively.

3. Integration with VIN/RC Data

Extend the system to fetch vehicle metadata through registration number or VIN (Vehicle Identification Number) using external APIs for automated and verified input.

4. User Feedback Loop

Implement a feedback module where users can rate prediction accuracy, allowing the system to learn and improve over time.

5. Cloud-Based API Services

Deploy the backend as a cloud-hosted API (using services like AWS Lambda or Azure Functions) for scalable integration with used-car dealer platforms.

REFERENCES

[1] K. Madhusudhanan et al., “Vehicle Price Prediction Using ML,” International Journal of Predictive Analytics, 2024.

[2] S. Roop and S. Tomar, “Car Damage Assessment with Deep Learning,” Journal of Computer Vision Applications, 2023.

[3] J. R. Krishnan and V. Selvaraj, “Predicting Resale Prices Using Ensemble Models,” AIP Conference Proceedings, 2022.

[4] P. Bharambe et al., “Vehicle Damage Detection Using CNN,” International Journal of Applied Sciences, 2022.

[5] T. Brown, “Image-Based Car Damage Detection Using Faster R-CNN,” IEEE Transactions on Neural Networks, 2021.

[6] F. R. Amik et al., “Assessing Damage Using Mask R-CNN,” Robotics and Automation Letters, 2021.

[7] A. Smith, “Leveraging CNN for Vehicle Damage Detection,” International Journal of Vision Research, 2020.

[8] E. Gegic et al., “Price Prediction of Used Cars Using ML,” TEM Journal, 2019.

[9] N. Pal, “Multi-Modal Data for Vehicle Price Prediction,” International Conference on Machine Learning, 2017.

[10] S. Pudaruth, “Predicting Used Car Prices Using ML,” International Journal of Information Technology, 2014.

[11] A. Kumar, J. Mehta, and R. Sharma, “Advancing Damage Detection in Vehicles with AI,” Journal of Artificial Intelligence Research, vol. 14, no. 3, pp. 132–145, 2023.

[12] M. Patel, K. Roy, and S. Khan, “Machine Learning for Used Car Price Prediction,” Applied AI Journal, vol. 9, no. 6, pp. 98–110, 2022.

[13] L. Williams, S. Brown, and C. Davis, “Integrating Image Analysis for Vehicle Pricing,” Pattern Recognition Letters, vol. 39, no. 1, pp. 245–260, 2021.

[14] R. K. Gupta, S. Chawla, and P. Verma, “Automating Damage Detection in Vehicles,” Transactions on Artificial Intelligence, vol. 18, no. 7, pp. 567–578, 2020.

[15] O. Lee, H. Kim, and Y. Park, “Predictive Analytics for Vehicle Resale Prices,” Advanced Machine Learning Review, vol. 5, no. 2, pp. 102–118, 2019.

[16] D. Zhang, L. Wang, and H. Li, “Used Car Price Estimation Based on Random Forest,” Journal of Data Mining Applications, vol. 6, no. 4, pp. 212–220, 2019.

[17] H. Chen and J. Liu, “Image-Based Defect Localization Using Deep Learning,” IEEE Access, vol. 8, pp. 55565–55574, 2020.

[18] V. Bhandari and S. Ghosh, “AI for Vehicle Appraisal: A Survey,” International Journal of Automotive AI, vol. 3, no. 1, pp. 14–25, 2021.

[19] A. R. Mehta and T. K. Jain, “Hybrid CNN-XGBoost for Vehicle Valuation,” Neural Computing and Applications, vol. 33, pp. 3121–3132, 2022.

[20] S. Agarwal, “Comparative Study of Regression Models in Car Pricing,” Data Science Review, vol. 11, no. 2, pp. 88–99, 2020.

[21] B. Thomas, K. Iyer, and A. Vyas, “Object Detection vs. Classification in Damage Detection,” Computer Vision Trends, vol. 9, no. 4, pp. 231–240, 2022.

[22] R. Sharma, N. Banerjee, and F. Ali, “Visual Inspection Using EfficientNet,” Journal of Deep Learning Research, vol. 7, no. 2, pp. 145–156, 2023.

[23] T. Nguyen, “Web-Based AI Tools for Automotive Applications,” Proceedings of the Smart Tech Symposium, vol. 2, no. 1, pp. 33–39, 2022.

APPENDIX I

Title of Project	AutoWorth: AI-Powered Car Resale Estimator
Implementation Details	AutoWorth is an AI-driven multimodal system designed to accurately predict the resale price of used vehicles by integrating both visual and tabular data. The system uses a Convolutional Neural Network (EfficientNet-B0) to detect surface-level damages from four car images and an XGBoost regression model to predict the resale price based on vehicle specifications such as brand, model, year, kilometers driven, fuel type, transmission, number of owners, and colour. The platform offers a transparent, condition-aware, and automated valuation tool, enhancing trust in online car resale markets.
Cost (hardware or software cost)	0 (developed using open-source tools and personal hardware resources)
Type	Application

Relevance		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO 10	PO 11	PO 12
	CO1	3	2	2	0	2	3	3	3	0	0	0	0
	CO2	3	3	3	3	2	3	3	2	0	0	3	0
	CO3	3	0	2	0	3	2	2	2	2	2	3	0
	CO4	0	0	0	0	1	0	0	1	3	3	0	0
Justification		Applies COCOMO for cost estimation.	Uses problem-solving in project.	Designs cost estimation models.	Applies research methods.	Uses advanced AI tools.	Utilizes modern tools.	Considers sustainability.	Follows ethical guidelines.	Collaborates effectively.	Communicates project details.	Manages project deliverables.	

Course outcomes	
CO1	Demonstrate the ability to synthesize and apply the knowledge and skills acquired in the academic program to real-world problems.
CO2	Evaluate different solutions based on economic and technical feasibility.
CO3	Effectively plan a project and confidently perform all aspects of project management.
CO4	Demonstrate effective written and oral communication skills.

PROGRAM OUTCOMES (PO's)

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

DEPARTMENT VISION:

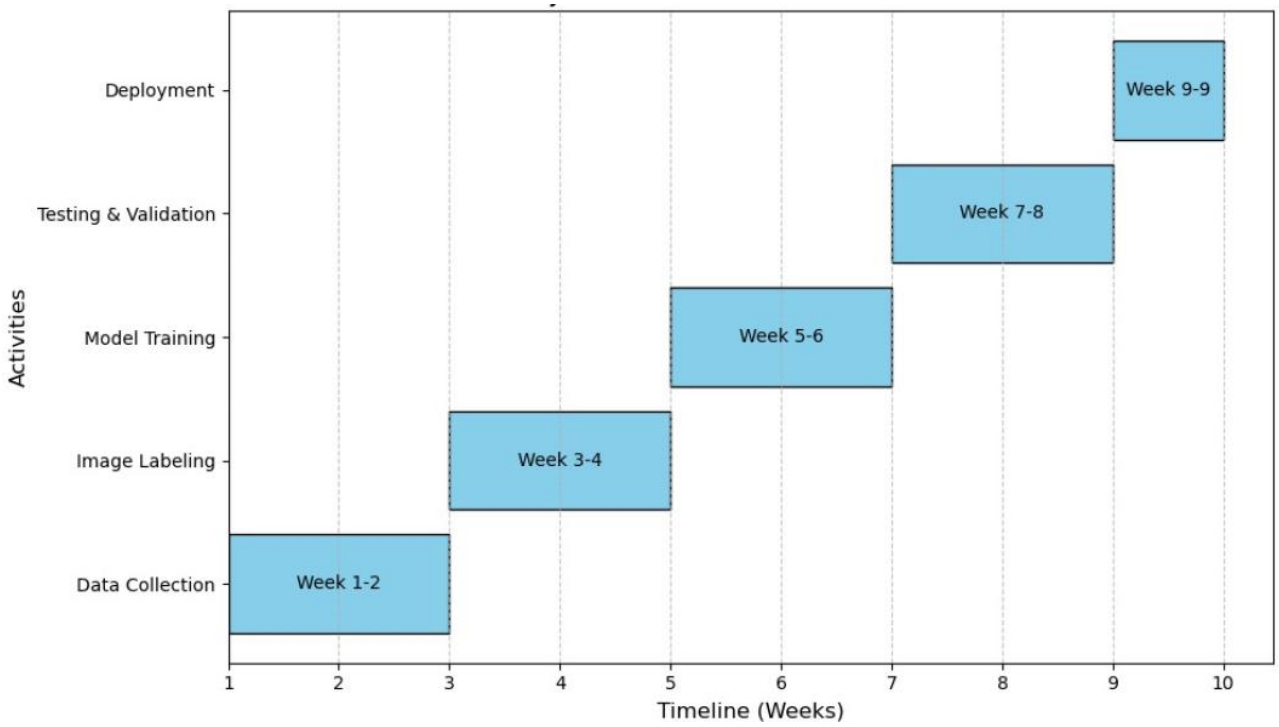
To contribute competent Computer Science and Artificial Intelligence professionals to the global talent pool to meet the constantly evolving societal needs.

DEPARTMENT MISSION:

Mentoring students towards a successful professional career in a global environment through quality education and soft skills in order to meet the evolving societal needs.

APPENDIX II

GANTT CHART



Activity	Plan Start (No of Weeks)	Plan Duration (No of Weeks)	Actual Start (No of Weeks)	Actual Duration (No of Weeks)	Project complete
Problem Definition	2	2	1	2	100%
Brainstorm Solutions	2	1	2	1	100%
Evaluate Solutions	3	1	3	3	100%
Prototype & Test Solutions	4	1	4	2	100%
Select Solutions	7	1	7	1	100%
Develop Solutions	8	2	8	2	100%
Implementation	9	3	9	2	100%
Train & Fine Tune Model	10	2	10	1	100%
Testing & Quality Assurance	11	2	11	2	100%
Deployment and Launch	12	2	12	1	100%